MODELLING, TRAJECTORY CALCULATION AND SERVOING OF A COMPUTER CONTROLLED ARM

Richard Paul

Stanford University

Prepared for:

Advanced Research Projects Agency

November 1972

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM-177

STAN-CS-72-311

MODELLING, TRAJECTORY CALCULATION AND SERVOING OF A COMPUTER CONTROLLED ARM

BY

RICHARD PAUL

SUPPORTED BY

ADVANCED RESEARCH PROJECTS AGENCY

ARPA ORDER NO. 457

NOVEMBER 1972

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-177

NOVEMBER 1972

COMPUTER SCIENCE DEPARTMENT
REPORT CS-311

MODELLING, TRAJECTORY CALCULATION AND SERVOING OF A COMPUTER CONTROLLED ARM

by

Richard Paul

The problem of computer control of an arm is divided into four parts: modelling, trajectory calculation, servoing and control.

In modelling we use a symbolic data structure to represent objects in the environment. The program considers how the hand may be positioned to grasp these objects and plans how to turn and position them in order to make various moves. An arm model is used to calculate the configuration-dependent dynamic properties of the arm before it is moved.

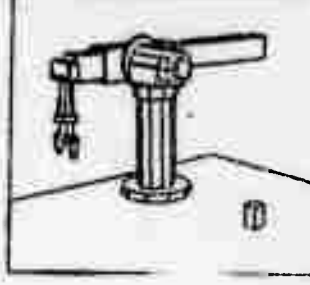
The arm is moved along time-coordinated space trajectories in which velocity and acceleration are controlled. Trajectories are calculated for motions along defined space curves, as in turning a crank; in such trajectories various joints must be free due to external motion constraints.

The arm is servoed by a small computer. No analog servo is used. The servo is compensated for gravity loading and for configuration-dependent dynamic properties of the arm.

In order to control the arm, a planning program interprets symbolic arm control instructions and generates a plan consisting of arm motions and hand actions.

The move planning program has worked successfully in the manipulation of plane faced objects. Complex motions, such as locating a bolt and screwing a nut onto it, have also been performed.

This research was supported in part by the Advanced Research Projects Agency of the Office of Defense under Contract No. SD-183.



ACKNOWLEDGMENTS

I wish to express my thanks to Professor Jerome Feldman for his invaluable help and advice.

I would like to thank Professor John McCarthy and Professor Bernard Roth for their suggestions.

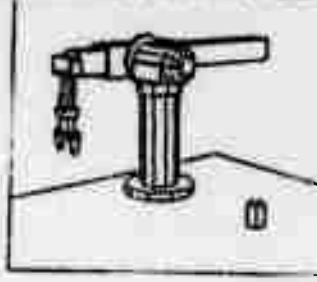
Many ideas presented in this work arose out of discussions with fellow workers, in particular with Gil Falk and Aharon Gill.

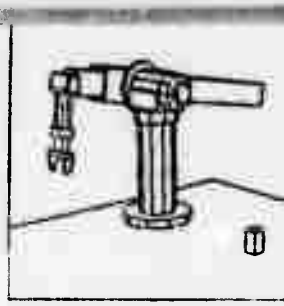
I would like to acknowledge the contributions from many people at the Stanford Artificial Intelligence Project and in particular from Victor Scheinman who designed the arm and Joe Zingheim who maintained it.

I am indebted to Bruce Baumgart who produced the small pictures of the arm appearing at the top corner of every page. By fanning through the document the example described in the text is displayed. These pictures were produced by Bruce using his GEOMED program [Baumgart] with an arm trajectory file as input.

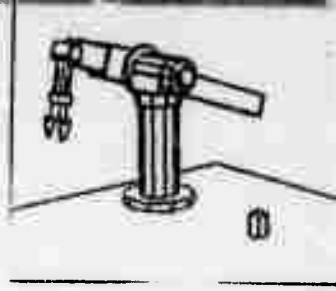
This work was submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of doctor of philosophy.

TABLE OF CONTENTS





LIST OF ILLUSTRATIONS



SECTION 1

INTRODUCTION

1.1 SYSTEM DESCRIPTION

We are concerned here with a computer controlled arm and hand. This arm and hand function together as a general purpose manipulator which forms part of a robot as an effector subsystem.

The computer plans and executes sequences of arm and hand motions to accomplish tasks. Arm motions are along smooth, collision free space trajectories; all known forces are predicted during planning and compensated for during execution. Hand motions, which consist of opening and closing the hand, are controlled by touch sensors. The hand can also exert a force while following an externally defined motion.

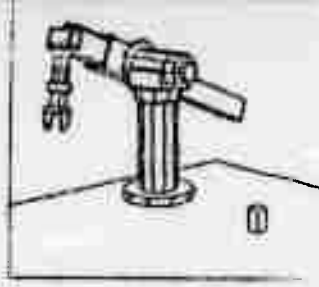
Any robot must contain both a sensory input mechanism, whereby it can gain information about the environment, and an effector subsystem by which it can change the environment. In addition to being the main effector, the hand also has primitive touch sensors and force detection ability, and may be considered a sensory subsystem.

The environment in which the robot works must be one which it can represent internally. It has certain information a priori, and is able to gain more information by interacting with its environment.

In our case it knows, a priori, that objects are plane-bounded, solid, and may be colored. It knows, a priori, that all objects are supported by a plane, or by each other, and that objects may be moved, but not through each other. It is given prototypes of all the possible objects that can exist, and learns by means of its senses of the existence and position of instances of these prototypes. Work is in progress to increase the scope of the robot's environment; curved objects are currently being added [Agin].

The type of interaction between robot subsystems is important as subsystems function together to accomplish tasks. Currently all subsystems perform under the direction of the strategy subsystem, with little interaction between other subsystems. For instance, the arm does not call for vision to locate an object that it has dropped, nor does vision call the arm to move an obscuring object in order that it may "see" better. One important exception is the case of visual feedback used to position the hand on an object; here the vision subsystem interacts directly with the arm [Gill].

To illustrate the system we will describe the interaction needed to solve the Instant Insanity puzzle [Feldman 71b]. Here it is first required that four colored cubes be found and the color of the faces be determined. The cubes are then turned and stacked so that each side of the stack has four different colored faces visible.



The interaction between the STRATEGY subsystem, VISION, RECOGNIZE, COLOR and the ARM is on the following level. The STRATEGY subsystem asks the VISION subsystem to find an outline; it then asks RECOGNIZE to identify the outline as a cube. This process is repeated until four cubes have been found. COLOR is then told to find the color at the center of each of the three visible faces of each cube. The ARM is told to turn over each cube and VISION called to refind the outline. RECOGNIZE is then called to reidentify the outline as a cube. When all four cubes have been turned over and re-found, COLOR is told to find the color of each of the three now visible back faces of each cube. The STRATEGY subsystem then determines the necessary turns and stack positions of each of the cubes to obtain a solution. The ARM is then told to move each cube accordingly. There are two levels of error recovery: within each subsystem, and by the strategy subsystem when a subsystem reports faiure.

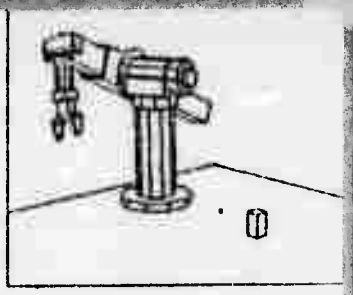
The programming environment in which the robot operates is known as the "Hand Eye" system [Feldman 71a]. Here individual subsystems are represented by time sharing jobs, which may interact. This is done by two means: 1) a global data structure which represents information about the environment available to all subsystems; 2) the message procedure construct whereby one job may execute a procedure in another job. For example, cubes are moved by a message procedure "MOVE_INSTANCE" which, in the Instant Insanity case, was used both to turn the cubes over and to stack them.

The arm subsystem provides a series of functions which are of general utility, such as to move the arm to a given position "MOVE," or to open the hand "OPEN." In order to perform these functions the arm subsystem maintains a data base in the form of constants and procedures which describe the arm and hand in detail. If the arm is required to perform some task then an attempt is made to describe the task in terms of existing arm functions, if this can be done then the problem is solved. If a task is proposed that cannot be solved in terms of existing functions, then either a function is modified or a new function must be written.

Other subsystems are not expected to use the arm's data base, which is local to the arm subsystem. Problem solving requiring use of this data base would be considered a function of the arm subsystem. Global variables are maintained which describe the current state of the arm for the other subsystems.

To build a stack of blocks would not be an arm function, as it could be accomplished in terms of simpler, existing functions such as MOVE_INSTANCE. To turn a cube over would be an arm function as this requires that the range throughout which the hand can grasp the cube be considered. To move the arm through a cluttered space of objects is an arm task as this requires that the arm be considered in relation to the other objects when planning the trajectory that the arm will follow.

Information is given to the arm subsystem in function calls and in the form of a three dimensional description of the space. If this space is completely described and the arm fails to accomplish a task then the strategy subsystem knows that the task must be specified differently. For instance, if a block must



be moved but some obstacle is in the way, then the strategy program must first ask the arm to move the obstruction and then ask that the original move be accomplished. The arm subsystem would not move other objects and thus change the state of the environment without being told to do so by the strategy subsystem. If the space model is incomplete and the arm cannot accomplish a task based on the available information, then the arm will not call the vision subsystem to have this space investigated but will report failure.

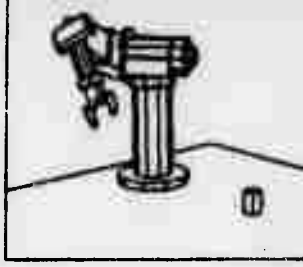
There are two main parts to the arm subsystem: the planning program and the arm execution program. In order to move the hand a plan is made. If the hand is to move an object then the planning program considers the object in relation to the hand to determine how it may be grasped. As the move is planned, other objects are considered in relation to the hand in order to prevent collisions. The plan is represented in terms of a coordinated time dependent trajectory for each joint of the arm. Since the computer has planned a trajectory the program knows before the arm is moved the configuration, velocity and acceleration of all the links of the arm and can compute the effective inertia and the gravity torque of each link. These terms together with the trajectory are given to the arm servo program, where the inertia and gravity terms are used to improve the execution of the trajectory.

The arm servo program executes the trajectory by moving the arm; it also performs such actions as opening and closing the hand. Trajectories, together with hand actions, are written out in a file, and may be repeatedly executed by the arm servo program if required. The arm servo program is small and is suitable for execution in a mini-computer connected directly to the arm. The planning program can be run under time sharing and can make plans for many such arms.

The servo is a conventional sampled data servo executed by the computer with the following modification: certain control constants, the loop gain, predicted gravity and external torques are varied with arm configuration.

In addition to the needs of the current vision and strategy subsystems, the arm has been programmed to perform other tasks such as turning cranks, screwing in screws, pushing and pulling. The touch sense is used in some of these tasks. Arm programs may be written where the course of execution of the program by the arm may be modified depending on activation of the touch sensors or other conditions. Such programs may be written in a form of assembly language but are identical to the message procedure calls of a strategy program.

In subsequent sections of this thesis we first describe the model of the arm and derive all the relations that we will use (Section 2). We then describe the model of the environment and the hand's interaction with it (Section 3). Section 4 describes MOVE_INSTANCE the highest level strategy function of the arm subsystem, provided primarily for strategy subsystems performing operations with plane faced solids. In Section 5 we describe the requirements and solution of the smooth trajectories used by the arm. The servo loop is described and we then deal with control(Section 7), giving a list of the arm functions or primitives.



The arm is described fully in [Scheinman], and we give a brief description in Appendix A.1. Two other appendices are given. Appendix A.2 describes SAIL, a form of ALGOL, with LEAP added. SAIL is the language in which the programs are written and we will use it to describe some of the algorithms. Appendix A.3 briefly describes homogeneous coordinate vectors and transformations, more fully covered in [Roberts 65].

The notation used in this work is as follows: vectors are represented by an underbar $\underline{V}$; Matrices are represented by vertical bars $|M|$; Exponents are either superscripted or preceeded by an "↑", x^2 or x↑2; Indices are either subscripted or enclosed in square brackets, A_{ij} or A[i,j]; Multiplication is represented by an asterisk "*".

1.2 HISTORY

The first mechanical hands were developed at Argonne National Laboratary in 1947 for handling radioactive materials [Goertz 64]. These hands were master-slave systems where the hand replicated the motions of a person, the master. In 1948 force feedback was added to enable the operator to feel the forces that the hand was exerting [Goertz 52].

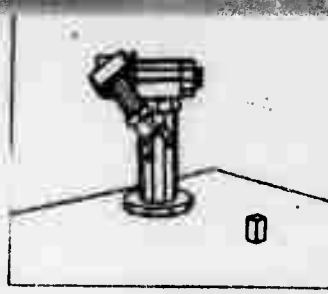
The early type of hand, without force feedback, has been adapted to perform repetitive tasks [Lindbom]. The hand is moved from one position to another by an operator and the joint positions recorded. The hand can then cycle repeatedly through these recorded positions in synchronism with external machinery.

In 1961 Ernst [Ernst] developed a computer controlled hand with touch sense. The hand could explore a region by touch and put the objects it identified into a box. In 1963 a half tone picture could be analyzed to locate and identify plane faced objects [Roberts 63]. By 1968 a program using a TV camera as vision input [Wichman] located the objects for the hand to pick up.

In 1968 Pieper studied the kinematics of arms and planned collision free trajectories through spaces containing obstacles [Pieper]. This work was followed by Kahn who studied the dynamics and developed a bang-bang servo [Kahn].

By adding force feedback Inoue was able to perform such tasks as putting a peg into a hole and turning a crank [Inoue]. Ejiri developed a system to assemble blocks using a drawing of the required assembly as a visual input [Ejiri]. In 1972 Goto could locate and identify plane faced objects by touch and then pack them compactly by moving and pushing them [Goto].

The recent proceedings of The Second International Symposium On Industrial Robots [IITRI] provide a general review of the state of the art.



SECTION 2

ARM MODEL

In this section we will consider the arm and develop a model for it [Pieper]. We will describe the solution which, given a hand position, returns a set of joint angles. Then, based on the model of the arm, we will develop a solution for differential motion. We will then derive the relation between acceleration and force for the arm, to obtain the effective link inertia and gravity loading [Kahn]. Finally we will derive the relationship between a force and moment acting at the hand and the six joint reaction torques. The results of this section are used in later sections of the work but as they all relate to the arm model they are derived together here.

2.1 KINEMATICS

The arm shown in Figure 2.1 is a six degree of freedom device allowing the hand to be positioned anywhere and with any orientation within the limits of joint motion.

The arm is made up of six links, each connected to the next by a joint. There are two kinds of joints, prismatic, or sliding, and revolute. In order to describe the link transformation in terms of the joint constraint and the joint variable we will introduce a coordinate system in which the joint constraint is implicit.

We will describe the "A" matrices, which relate between link coordinate systems, and the "T" matrices, the link transformations which specify the position and orientation of each link in space.

Associated with each link is an orthogonal coordinate system fixed in the link (see Figure 2.2).

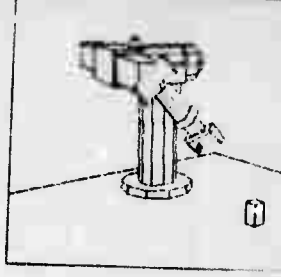
For link i the Zi axis is directed along the axis of the joint between link i and i+1. The xi axis is along the common normal between the two joint axes of the link in the direction from z(i-1) to zi. The y axis completes the right handed set.

We can transform coordinate systems i into i+1 by performing a rotation, two translations, and a final rotation as follows:

1). A rotation about zi of θi to make xi parallel to x(i+1).

2). A translation si along zi to locate the origin at the point where the common normal between zi and zi+1 cuts zi.

3). A translation of ai along x(i+1) to bring the origins into coincidence.



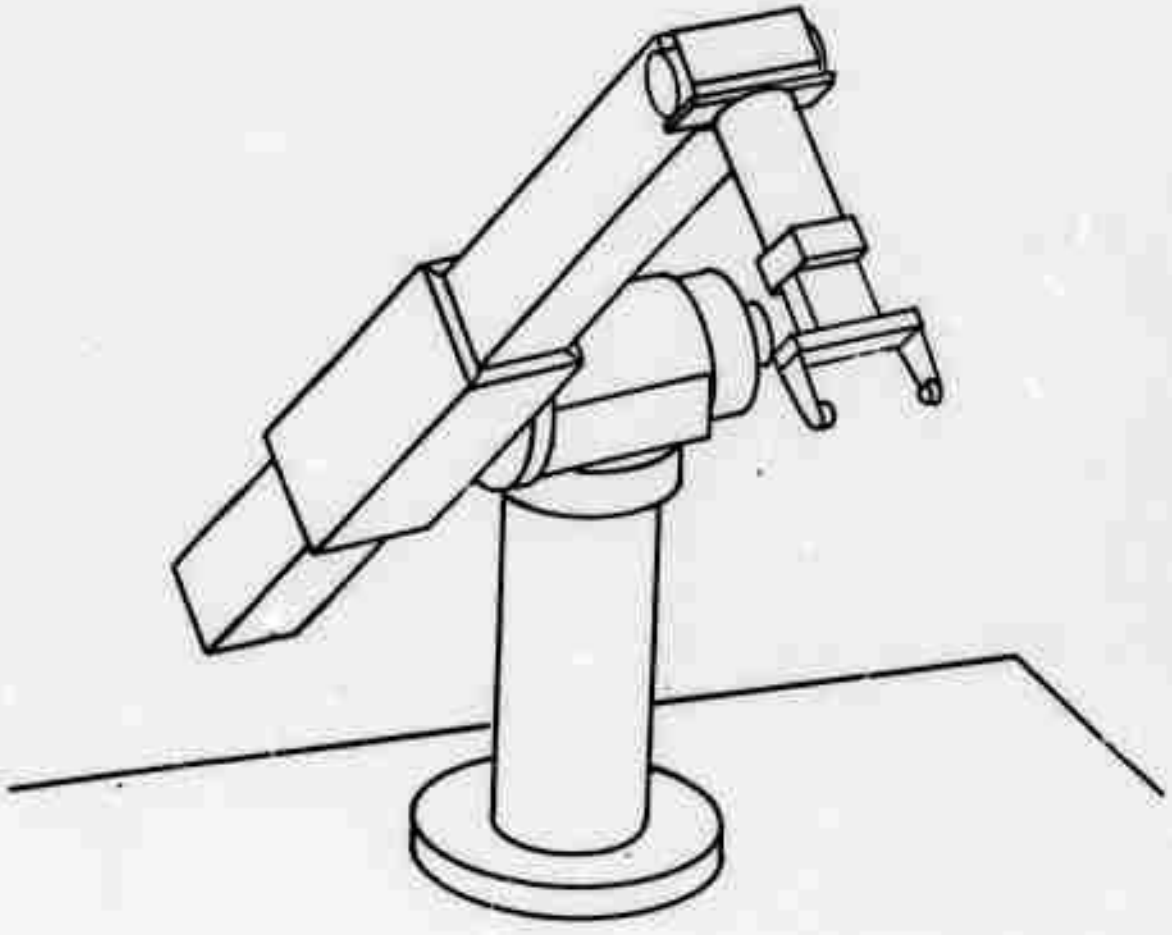
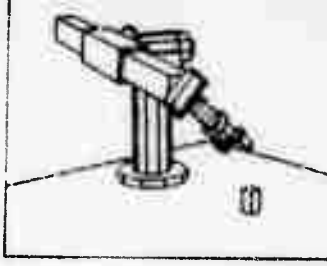


Figure 2.1
Arm



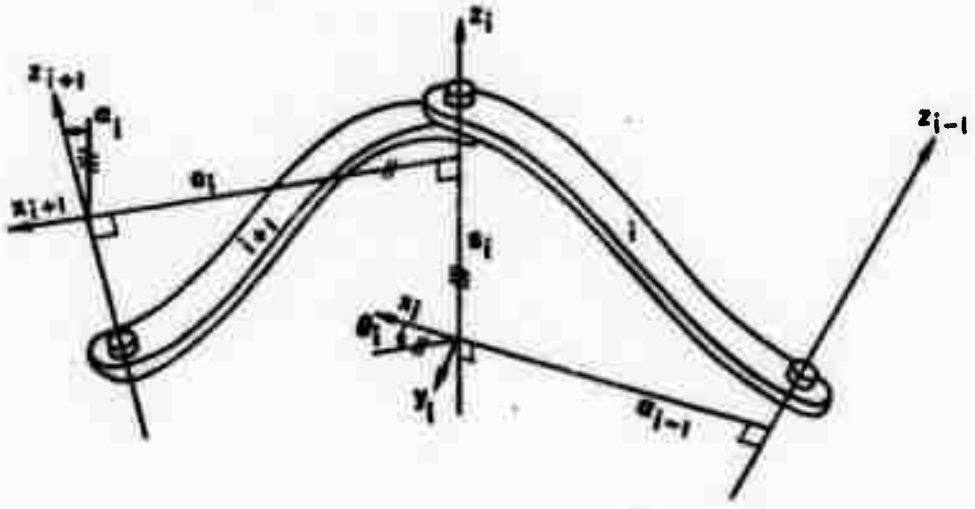


Figure 2.2
Link Coordinate System

4). A rotation about x(i+1) of αi to bring the z axes into coincidence.

The joint variable for a revolute joint is θ; the joint variable for a prismatic joint is s.

In the case of the arm that we are using, we pick the origin of coordinates at the base of the shoulder. We have two revolute joints, followed by a prismatic joint, followed by three intersecting revolute joints. There are three offsets: S1 = 16.24in, S2 = 6.05in and S6 = 10.35in. S4 = S5 = 0, and S3 is a variable. The link coordinates for the arm are shown in Figure 2.3.

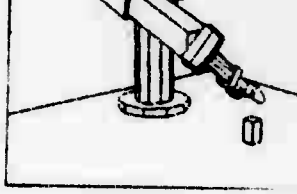
If we express points in link i by a vector Ri then the relationship between coordinate systems Ri and R(i-1) may be expressed by:

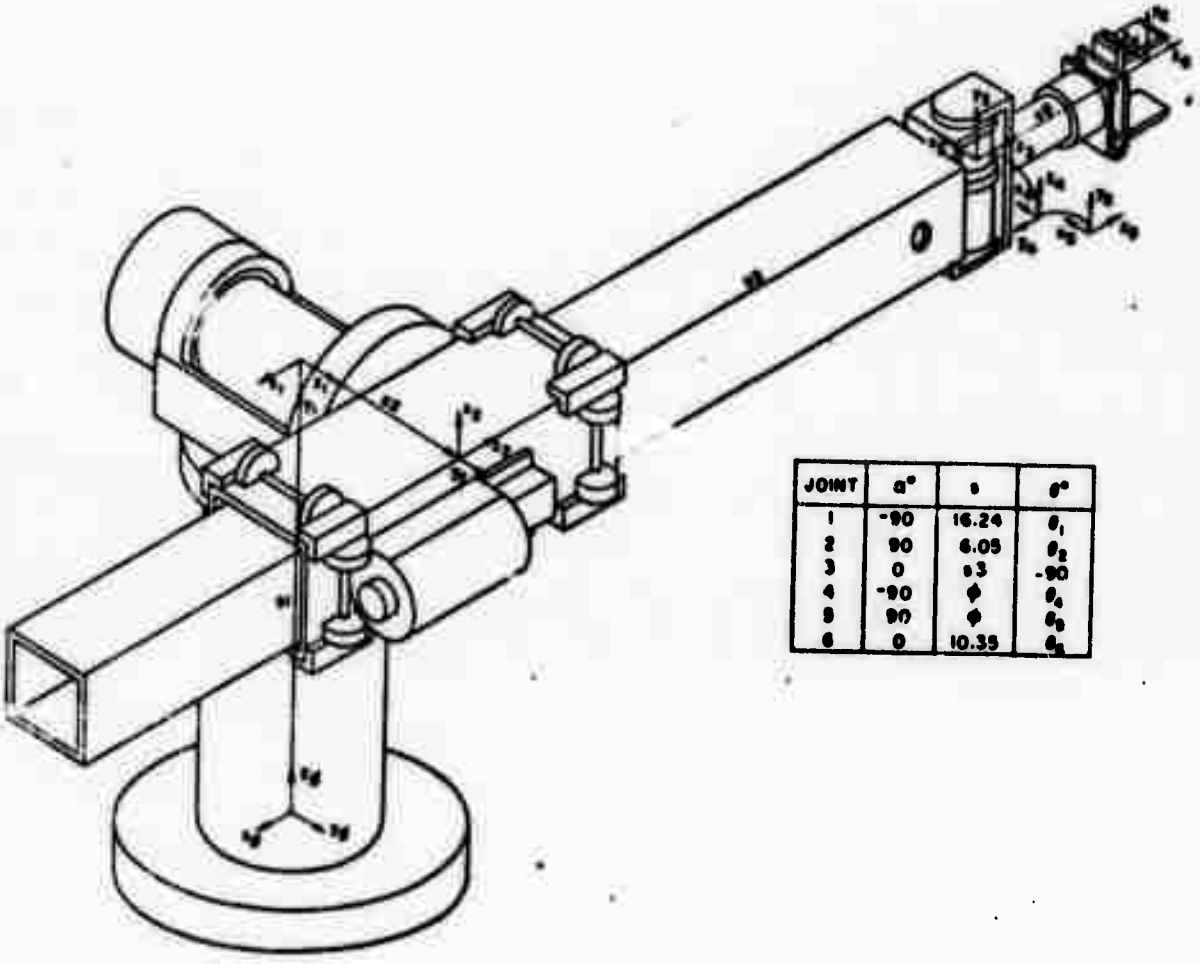
$$R(i-1) = |Ai| * |Ri| \qquad \text{[Eq. 2.1]}$$

where | Ai | is given by:

$$\begin{vmatrix} \cos\theta & -\cos\alpha\sin\theta & \sin\alpha\sin\theta & a\cos\theta \\ \sin\theta & \cos\alpha\cos\theta & -\sin\alpha\cos\theta & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & s \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

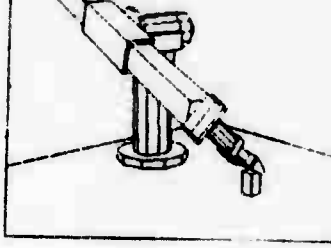
[Eq. 2.2]





JOINT	α°	a	θ°
1	-90	16.24	θ_1
2	90	6.05	θ_2
3	0	13	-90
4	-90	ϕ	θ_4
5	90	ϕ	θ_5
6	0	10.35	θ_6

Figure 2.3
Arm Coordinate Systems



For the joint angles shown in the following table:

Table 2.1

Joint Angles

JOINT	VARIABLE
1	-95.7
2	-112.4
3	22.2
4	-38.2
5	80.4
6	68.9

The A matrices for the arm shown in Figure 2.1 are:

$$A1 = \begin{vmatrix} -.10 & .00 & 1.00 & .00 \\ -1.00 & .00 & -.10 & .00 \\ .00 & -1.00 & .00 & 16.24 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$A2 = \begin{vmatrix} -.38 & .00 & -.92 & .00 \\ -.92 & .00 & .38 & .00 \\ .00 & 1.00 & .00 & 6.05 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$A3 = \begin{vmatrix} .00 & 1.00 & .00 & .00 \\ -1.00 & .00 & .00 & .00 \\ .00 & .00 & 1.00 & 22.16 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$A4 = \begin{vmatrix} .79 & .00 & .62 & .00 \\ -.62 & .00 & .79 & .00 \\ .00 & -1.00 & .00 & .00 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$A5 = \begin{vmatrix} .17 & .00 & .99 & .00 \\ .99 & .00 & -.17 & .00 \\ .00 & 1.00 & .00 & .00 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$A6 = \begin{vmatrix} .36 & -.93 & .00 & .00 \\ .93 & .36 & .00 & .00 \\ .00 & .00 & 1.00 & 10.35 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

if we let link 0 be the table coordinate system then we may relate from any link coordinates to R0 by:

$$R0 = |A1|*|A2|*|A3| \ ... \ |Ai|*|Ri| \qquad \text{[Eq. 2.3]}$$

or:

$$R0 = |Ti| * |Ri| \qquad \text{[Eq. 2.4]}$$

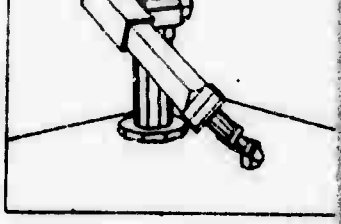
where:

$$|Ti| = |A1|*|A2|*|A3| \ ... \ |Ai| \qquad \text{[Eq. 2.5]}$$

| Ti | is the transform of the ith link of the arm which describes the position of the link in table coordinates.

For the A matrices given in the preceding example we have the corresponding T matrices:

$$T1 = \begin{vmatrix} -.10 & .00 & 1.00 & .00 \\ -1.00 & .00 & -.10 & .00 \\ .00 & -1.00 & .00 & 16.24 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$



$$T2 = \begin{vmatrix} .04 & 1.00 & .09 & 6.02 \\ .38 & -.10 & .92 & -.60 \\ .92 & .00 & -.38 & 16.24 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

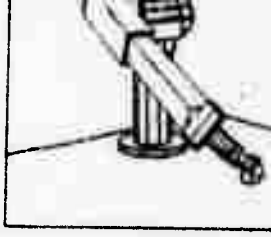
$$T3 = \begin{vmatrix} -1.00 & .04 & .09 & 8.06 \\ .10 & .38 & .92 & 19.78 \\ .00 & .92 & -.38 & 7.79 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$T4 = \begin{vmatrix} -.81 & -.09 & -.59 & 8.06 \\ -.16 & -.92 & .36 & 19.78 \\ -.57 & .38 & .73 & 7.79 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$T5 = \begin{vmatrix} -.23 & -.59 & -.78 & 8.06 \\ -.93 & .36 & -.00 & 19.78 \\ .28 & .73 & -.63 & 7.79 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

$$T6 = \begin{vmatrix} -.63 & -.00 & -.78 & -.00 \\ .00 & 1.00 & -.00 & 19.78 \\ .78 & -.00 & -.63 & 1.30 \\ .00 & .00 & .00 & 1.00 \end{vmatrix}$$

| T6 | is the transformation of the hand, the last link of the arm. We can interpret this matrix as follows: the right hand column of the matrix is the position, in table coordinates, of a point centrally located between the finger tips P. The second column specifies the direction of the "y" axis of the hand (see Figure 2.3), which we will call the orientation vector O; the orientation vector is directed between the finger tips. The third column is the "z" axis and is in the direction that the hand is pointing; we will call this vector the approach vector A. We may then write | T6 | as:



$$\begin{vmatrix} (O \times A)[x] & O[x] & A[x] & P[x] \\ (O \times A)[y] & O[y] & A[y] & P[y] \\ (O \times A)[z] & O[z] & A[z] & P[z] \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \text{[Eq. 2.6]}$$

The approach vector can be expressed in terms of an approach angle as follows:

We define a reference approach vector $\underline{RA}$ as:

$$\underline{RA} \leftarrow \underline{O} \times \underline{k} \qquad \text{[Eq. 2.7]}$$

where $\underline{k}$ is a unit z vector. The approach angle is the angle between $\underline{A}$ and $\underline{RA}$ measured about $\underline{O}$ (see Figure 2.4).

2.2 ARM SOLUTION

The arm solution is a procedure which, given | T6 |, the transformation of the hand, returns the six joint angles which will position the arm in such a way that the hand will have the required transformation.

Because the last three joints intersect we can obtain a closed form solution [Pieper]. The position of the end of link 3 is found as:

$$\underline{L3} \leftarrow \underline{P} - \underline{L5} \qquad \text{[Eq. 2.8]}$$

where $\underline{L5}$ is a vector the length of S6 and in the direction of the approach vector $\underline{A}$ (see Figure 2.5).

A vector from the shoulder to the end of link 3 is:

$$\underline{W} \leftarrow \underline{L3} - \underline{L1} \qquad \text{[Eq. 2.9]}$$

We can now solve for S3, the prismatic joint variable, as:

$$S3 = (\underline{W} \cdot \underline{W} - S2\uparrow 2)\uparrow(1/2) \qquad \text{[Eq. 2.10]}$$

Having solved for S3, $\theta 1$ is given by:

$$\theta 1 = \theta + \phi \qquad \text{[Eq. 2.11]}$$

Where θ and ϕ are:

SOLUTION

$$\text{Tan } \theta = W[2] \; / \; W[1] \qquad \text{[Eq. 2.12]}$$

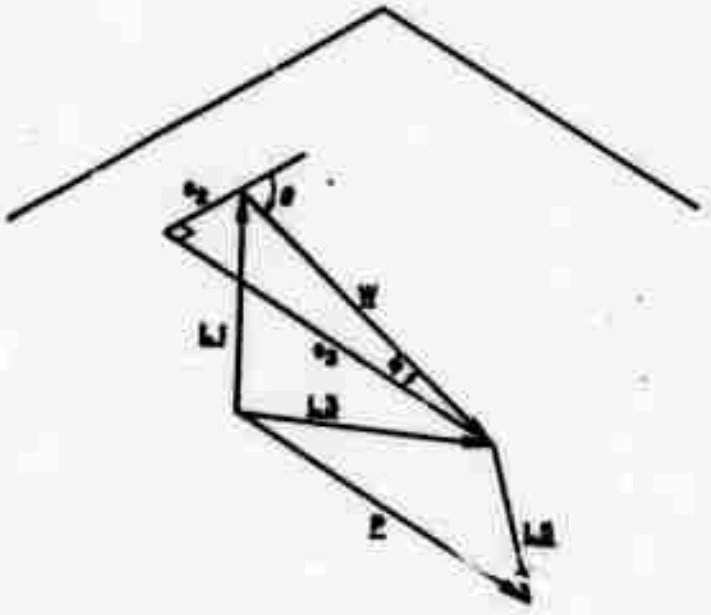


Figure 2.5
Solution of Joints 1,2 and 3

$$\text{Sin } \Phi = S2 \; / \; (\; W[1]\uparrow 2 + W[2]\uparrow 2 \;)\uparrow(1/2) \qquad \text{[Eq. 2.13]}$$

See Figure 2.5

$\theta 2$ is then:

$$\text{Cos } \theta 2 = W[3] \; / \; S3 \qquad \text{[Eq. 2.14]}$$

See Figure 2.5.

The unit vectors y3 and z3 are now calculated for link 3 and a vector:

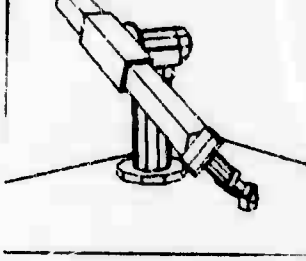
$$\underline{RR} = \underline{z3} \times \underline{z6} \qquad \text{[Eq. 2.15]}$$

is calculated.

Then:

$\theta 4$	angle between $\underline{RR}$ and $\underline{u3}$ about $\underline{z3}$	[Eq. 2.16]
$\theta 5$	angle between $\underline{z6}$ and $\underline{z3}$ about $\underline{RR}$	[Eq. 2.17]
$\theta 6$	angle between $\underline{u6}$ and $\underline{RR}$ about $\underline{z6}$	[Eq. 2.18]

See Figure 2.6



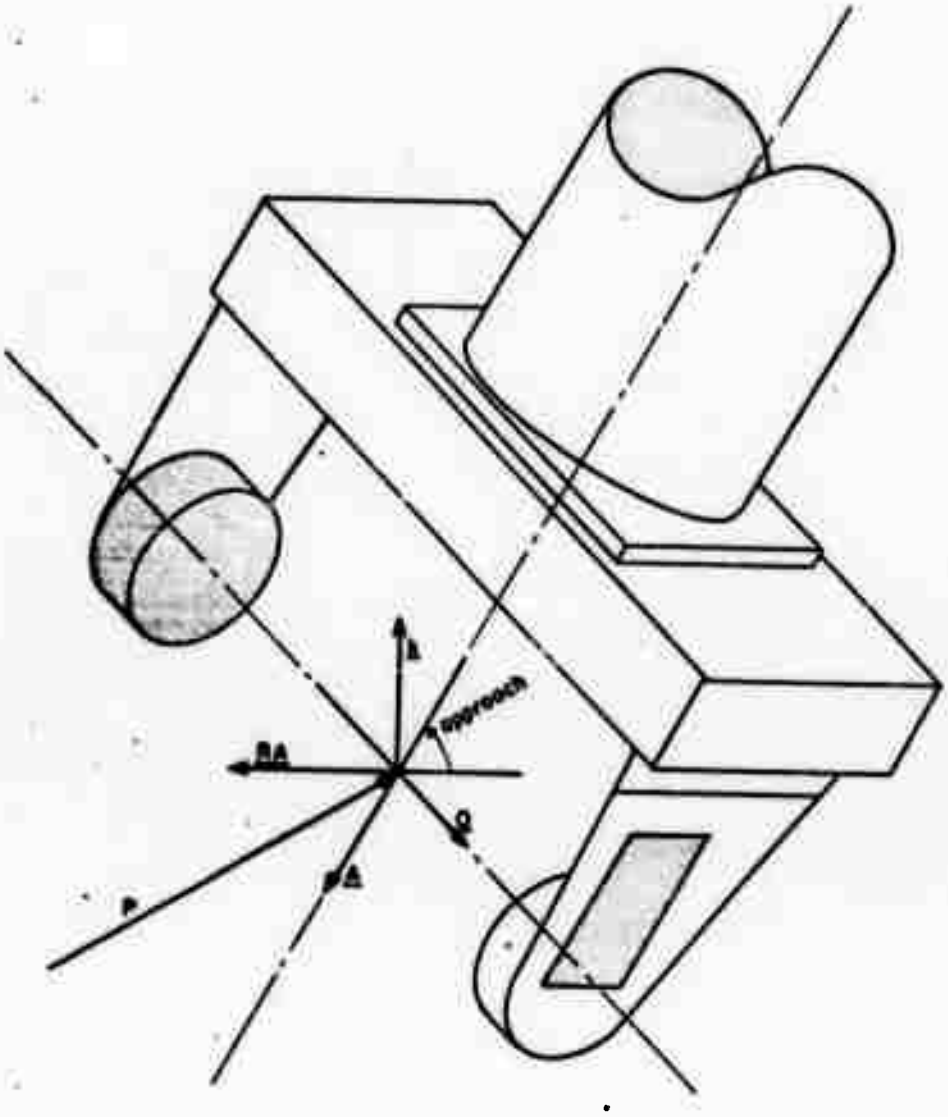


Figure 2.4
Hand Coordinates

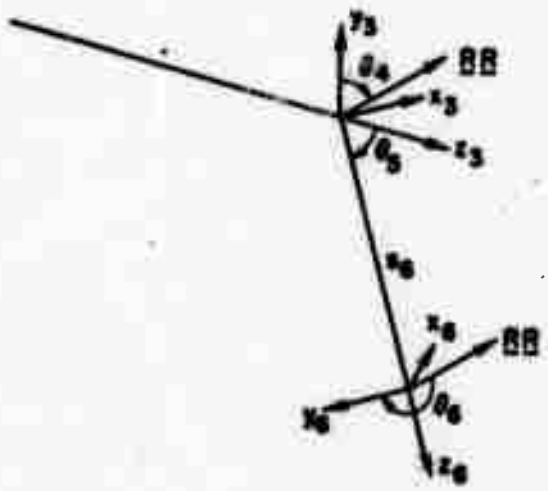


Figure 2.6
Solution of Joints 4,5 and 6

All joints with the exception of joint 6 have only a partial range of motion. As the solution for each joint is obtained, it must be checked to see that it is within the range of motion of that joint.

2.3 DIFFERENTIAL SOLUTION

Given an arm solution, it is often necessary to compute the differential change in joint angles in order to make a small change in position while maintaining the current orientation of the hand.

We can obtain the differential change in position $\underline{dR\emptyset}$ with respect to a change in joint variable dqj as:

$$\underline{dR\emptyset} = \sum_{j=1}^{i} |Uij| * dqj * |Ri| \qquad \text{[Eq. 2.19]}$$

where:

$$|Uij| = \partial|Ti|/\partial qj \qquad \text{[Eq. 2.20]}$$

From which we obtain:

$$|Uij| = |A1|*|A2| \ldots |Aj-1|*|Qj|*|Aj|*|Aj+1| \ldots *|Ai| \qquad \text{[Eq. 2.21]}$$

and depending on whether the joint is rotary:

$$Q(\theta) = \begin{vmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix} \qquad \text{[Eq. 2.22]}$$

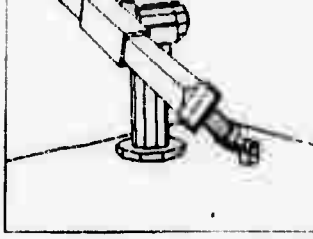
or prismatic:

$$Q(s) = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix} \qquad \text{[Eq. 2.23]}$$

If we then evaluate the six | U6j| matrices (Equation 2.20) we will have the differential change in the 16 elements of the hand matrix | T6 | for each of the six joints. This corresponds to 16 equations in 6 unknowns of which only 6 equations are independent. We must pick 6 independent equations of the 16 and then solve for the required change in the qj.

From | U6j | we pick the first three elements of the right hand column as these correspond to dx, dy, dz. We then pick the two smallest elements of column 3, the approach vector, to constrain its direction. Finally we pick one additional element from column 2, the orientation vector, in order to constrain rotation about the approach vector.

The six equations:



$$\begin{vmatrix} dx \\ \\ dy \\ \\ dz \\ \\ \theta \\ \\ \theta \\ \\ \theta \end{vmatrix} = \begin{vmatrix} U61_{14} & U62_{14} & U63_{14} & U64_{14} & U65_{14} & U66_{14} \\ \\ U61_{24} & U62_{24} & U63_{24} & U64_{24} & U65_{24} & U66_{24} \\ \\ U61_{34} & U62_{34} & U63_{34} & U64_{34} & U65_{34} & U66_{34} \\ \\ U61_{u3} & U62_{u3} & U63_{u3} & U64_{u3} & U65_{u3} & U66_{u3} \\ \\ U61_{v3} & U62_{v3} & U63_{v3} & U64_{v3} & U65_{v3} & U66_{v3} \\ \\ U61_{w2} & U62_{w2} & U63_{w2} & U64_{w2} & U65_{w2} & U66_{w2} \end{vmatrix} * \begin{vmatrix} dq_1 \\ \\ dq_2 \\ \\ dq_3 \\ \\ dq_3 \\ \\ dq_4 \\ \\ dq_6 \end{vmatrix}$$

[Eq. 2.24]

are then solved to give the six dqj, the differential change in joint angle.

For the position that we have been considering and have defined the A and T matrices. A differential change of dz = 3.0 in is found to be:

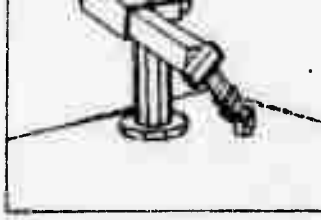
Table 2.2

Differential Change of Joint Angle

JOINT	dq
1	0.0
2	7.2
3	-1.1
4	-1.0
5	4.4
6	5.7

2.4 DYNAMICS

From the kinematic arm model we can also develop the dynamic model [Uicker]. We will derive the Lagrangian [Kahn] for the arm in a gravitational force field, and obtain the equations relating acceleration to joint torque, including the static torques necessary to overcome the effects of gravity. These results will be used in the section which relates to servoing the arm.



If the Lagrangian L is defined as:

$$L = K - P \qquad \text{[Eq. 2.25]}$$

where:

K is the Kinetic Energy of the system in terms of joint variables;

P is the potential energy in terms of joint variables.

The joint variables "q" are either "θ" or "s" depending upon whether the joint is revolute or prismatic.

The equations of motion are given by:

$$d/dt \left(\partial L/\partial \dot{q}j \right) - \partial L/\partial qj = Fj \qquad \text{[Eq. 2.26]}$$

$$\text{for } j = 1,2, \ldots n$$

where Fj is the force on joint j.

From Equation 2.19 we can obtain the velocity of any point as:

$$d\underline{R0}/dt = | Vi | * | Ri | \qquad \text{[Eq. 2.27]}$$

where:

$$| Vi | = \sum_{j=1}^{i} (| Uij | * dqj/dt) \qquad \text{[Eq. 2.28]}$$

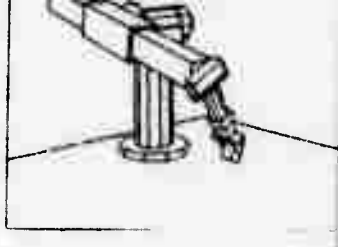
We may now express the kinetic energy of a link as follows. Consider a particle of mass dm on link i at $\underline{Ri}$ then the kinetic energy is:

$$dHi = 1/2 \left(\dot{\underline{Ri}} \cdot \dot{\underline{Ri}} \right) dm \qquad \text{[Eq. 2.29]}$$

or:

$$dHi = 1/2 \, \text{Trace}(| Vi | * | Ri | * | Ri |^{T} * | Vi |^{T}) * dm \qquad \text{[Eq. 2.30]}$$

The total kinetic energy for the link can be found by integrating over the mass of the link.



$$dHi = 1/2\ \mathrm{Trace}\left[|Vi|*\left(\int_{link} |Ri|*|Ri|^{T} *dm\right)*|Vi|^{T}\right]$$

[Eq. 2.31]

to obtain:

$$Hi = mi* \begin{vmatrix} 1/2(-k_{i11}^2+k_{i22}^2+k_{i33}^2) & k_{i12}^2 & k_{i13}^2 & \bar{x}_i \\ k_{i12}^2 & k_{i11}^2-k_{i22}^2+k_{i33}^2 & k_{i23}^2 & \bar{y}_i \\ k_{i13}^2 & k_{i23}^2 & k_{i11}^2+k_{i22}^2-k_{i33}^2 & \bar{z}_i \\ \bar{x}_i & \bar{y}_i & \bar{z}_i & 1 \end{vmatrix}$$

[Eq. 2.32]

where: k_{ijk} is the radius of gyration of link i about the j,k axes. $\bar{x}i$, $\bar{y}i$, $\bar{z}i$ is the center of mass of link i. mi is the mass of link i.

The kinetic energy of the entire system is:

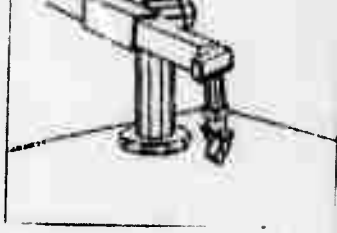
$$K = 1/2 \sum_{i=1}^{n} \mathrm{Trace}(|Vi|*|Hi|*|Vi|^{T}) \qquad \text{[Eq. 2.33]}$$

The potential energy of the system due to gravity in the negative z direction is expressed by:

$$P = -\sum_{i=1}^{n} mi * |G| * |Ti| * |\bar{R}i| \qquad \text{[Eq. 2.34]}$$

where: $|G| = |0\ \ 0\ \ g\ \ 0|$ [Eq. 2.35]

and g is the acceleration due to gravity.



Substituting for K from Equation 2.33 and for P from Equation 2.34 into Equation 2.25 and then differentiating according to Equation 2.26 we obtain:

$$Fi = \sum_{j=i}^{n} \sum_{k=1}^{j} \mathrm{Trace}(|Ujk|*|Hj|*|Uji|^{T} *\ddot{q}k)$$

$$+\sum_{j=i}^{n} \sum_{k=1}^{j} \sum_{p=1}^{j} \mathrm{Trace}(|Ujkp|*|Hj|*|Uji|^{T} *\dot{q}j*\dot{q}k)$$

$$-\sum_{j=i}^{n} mj*|G|*|Uji|*\bar{R}j \qquad \text{[Eq. 2.36]}$$

the equation relating acceleration to force, from which we will infer the effective link inertia and gravity loading torque Tg in Subsection 6.1.

2.5 FORCE AND MOMENTS

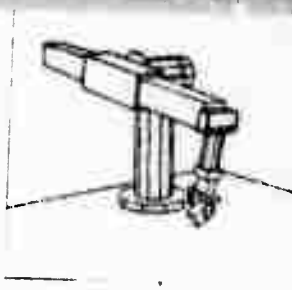
Given a force $\underline{F}$ that acts through the origin of the hand coordinate system and a moment M, we wish to find the joint reaction torques.

We represent a force in the n'th coordinate system as:

$$Fn = \begin{vmatrix} Fn[x] \\ Fn[y] \\ Fn[z] \\ 0 \end{vmatrix} \qquad \text{[Eq. 2.37]}$$

and a moment similarly:

$$Mn = \begin{vmatrix} Mn[x] \\ Mn[y] \\ Mn[z] \\ 0 \end{vmatrix} \qquad \text{[Eq. 2.38]}$$



We first transform the force F and the moment M into the hand coordinate system by:

$$| F6 | = | T6 |^{-1} * | F | \qquad \text{[Eq. 2.39]}$$

$$| M6 | = | T6 |^{-1} * | M | \qquad \text{[Eq. 2.40]}$$

We then proceed to transform the force and moment back through the links as follows (see Figure 2.7):

$$| F(n-1) | = | An | * | Fn | \qquad \text{[Eq. 2.41]}$$

and the moment:

$$| M(n-1) | = | An | * | Mn | \qquad \text{[Eq. 2.42]}$$

However the force F(n-1) does not act through the origin of the (n-1)th coordinate system (see Figure 2.7) and we have an additional moment given by:

$$\underline{P} \times \underline{F(n-1)} \qquad \text{[Eq. 2.43]}$$

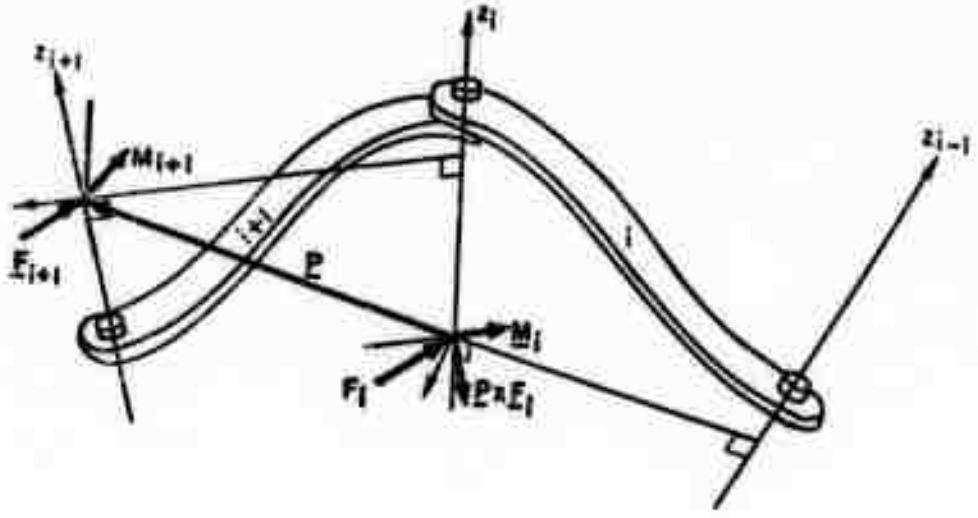
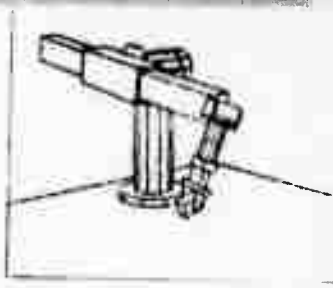


Figure 2.7
Force transformation

where: $\underline{P}$ is the right hand column of | An |



the total moment is then:

$$| M(n-1) | = | An | * | Mn | + \underline{P} \times \underline{F(n-1)} \qquad \text{[Eq. 2.44]}$$

If the (n-1)th joint is revolute then the reaction torque is M(n-1)[z]; if it is prismatic then the force is F(n-1)[z].

The six reaction torques are known as the equivalent arm torque.

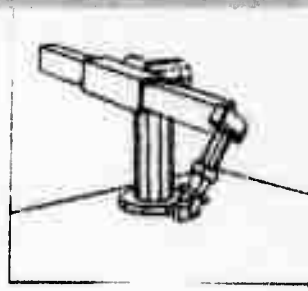
When the hand is to exert a force the equivalent arm torque is calculated and added to the gravity torque for each joint. If each joint is run at these torques then the hand will exert the required force.

For the arm position we have been considering we have two examples, the first is to exert a force of -100oz. in the z direction, the second example is to exert a moment of -100oz. in. about the z axis:

Table 2.3

Equivalent Arm Torques

F[z]=-100oz.		M[z]=-100oz. in.	
JOINT	TORQUE	JOINT	TORQUE
1	0.0	1	-100.0
2	-1968.0	2	0.0
3	38.2	3	0.0
4	-741.9	4	38.2
5	-290.6	5	-72.7
6	0.0	6	62.7



SECTION 3

WORLD MODEL

This section describes the model of the arm's environment, which consists of solid plane-faced objects. These are the objects that the vision system can identify [Falk] and a representation of them is maintained. This is done for both the Arm and the Vision programs, which share a common data base [Paul].

We will first describe the prototype representation and the manner of specifying instances of these prototypes. The problem of grasping this class of objects is then reduced to finding a set of orientation vectors.

3.1 PROTOTYPE DESCRIPTION

Objects are described in terms of prototypes. To identify an object is to associate the object with its prototype; one prototype can represent many objects or instances. All common information relating to the instances is kept only once, with the prototype. The position of an object is associated with the instance, as every instance has a different position. The prototype is located with its center of mass at the origin and its principal inertial axes aligned with the coordinate axes. Each vertex is represented by a vector giving its distance from the origin and each face is represented by a row matrix giving its position and outward pointing normal.

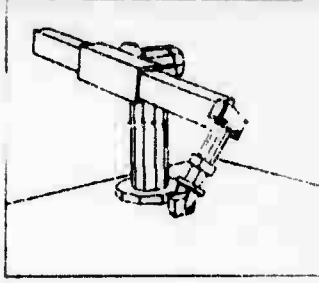
Items are created for each part of a prototype (face, vertex, edge). (Readers not familiar with "items" should consult Appendix A.2.) The spatial information associated with vertices and faces is stored as array datums of these items. Each of these items is associated with the prototype, which is itself an item. Different attributes are used to indicate which topological part is being associated.

For example, in the case of the cube shown in Figure 3.1:

```
FACE ⊗ CUBE ≡ F1
FACE ⊗ CUBE ≡ F2                                   [Eq. 3.1]
  "      "     "
FACE ⊗ CUBE ≡ F6

VERTEX ⊗ CUBE ≡ V1
VERTEX ⊗ CUBE ≡ V2                                 [Eq. 3.2]
  "        "     "
VERTEX ⊗ CUBE ≡ V8

EDGE ⊗ CUBE ≡ E1
EDGE ⊗ CUBE ≡ E2                                   [Eq. 3.3]
  "      "     "
EDGE ⊗ CUBE ≡ E12
```



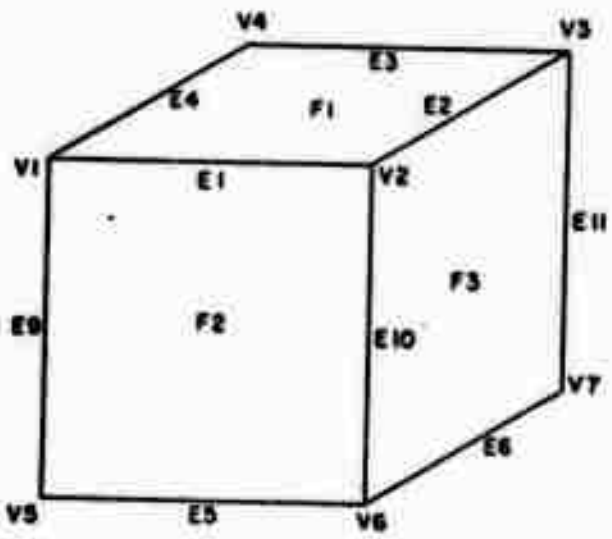


Figure 3.1
Cube Prototype

In the case of edges, the length of the edge is kept as its datum.

For each Face we now associate its vertices and edges together

Consider for example face F1:

$$\begin{aligned} &\text{BOUNDARY} \circledcirc F1 \equiv E1 \\ &\text{BOUNDARY} \circledcirc F1 \equiv E2 \\ &\text{BOUNDARY} \circledcirc F1 \equiv E3 \\ &\text{BOUNDARY} \circledcirc F1 \equiv E4 \end{aligned} \qquad \text{[Eq. 3.4]}$$

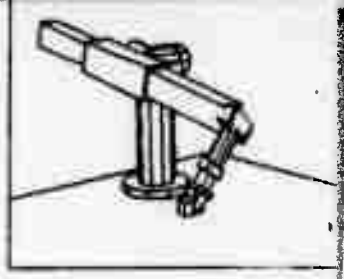
$$\begin{aligned} &\text{CORNER} \circledcirc F1 \equiv V1 \\ &\text{CORNER} \circledcirc F1 \equiv V2 \\ &\text{CORNER} \circledcirc F1 \equiv V3 \\ &\text{CORNER} \circledcirc F1 \equiv V4 \end{aligned} \qquad \text{[Eq. 3.5]}$$

And for edges we associate the edge with its endpoints:

$$\begin{aligned} &\text{ENDPT} \circledcirc E1 \equiv V1 \\ &\text{ENDPT} \circledcirc E1 \equiv V2 \end{aligned} \qquad \text{[Eq. 3.6]}$$

etcetera.

Prototypes are kept for the objects shown in Figure 3.2, this data is kept in the global data store and is available to all programs.



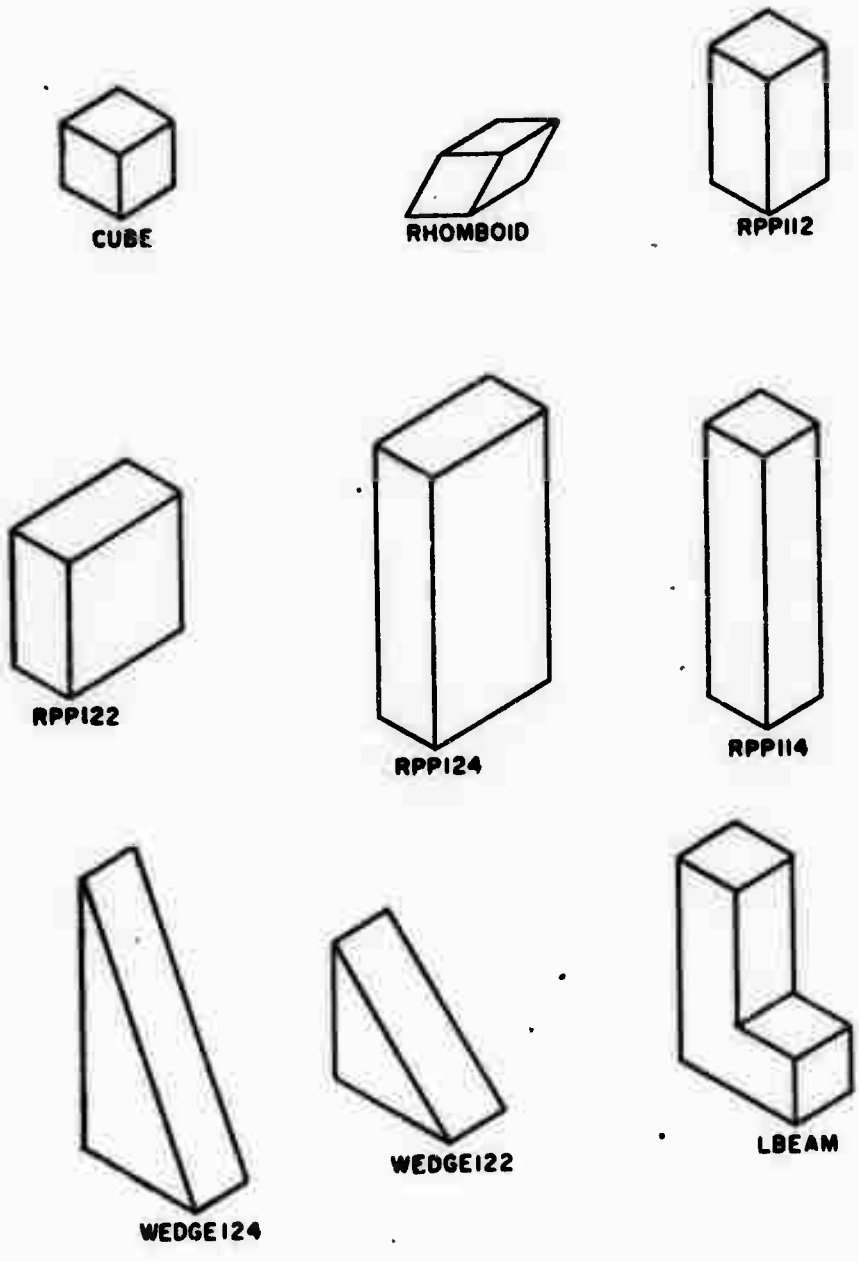
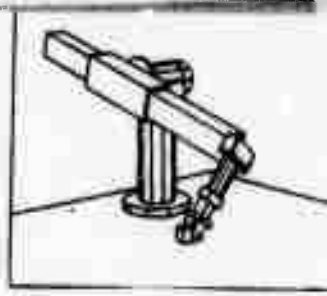


Figure 3.2
Prototypes



Based on such a description it is possible to find many relationships, for example:

Given any face F1 find its neighbors which share a common vertex V1.

We can find the faces which share the common vertex V1 by finding all the faces which satisfy

```
FOREACH F | VERTEX ⊛ F ≡ V1
```

Here F represents an item variable satisfied by the association. The set of these F's are all the faces which share this common vertex. However they are not all neighbors of F1. To ascertain that they are neighbors we must require that they share a common edge and that they are not F1.

```
FOREACH F,E | VERTEX ⊛ F ≡ V1
        ∧   EDGE ⊛ F  ≡ E
        ∧   EDGE ⊛ F1 ≡ E
        ∧   F ≠ F1
```

This specifies the set of faces F which are the neighbors of F1 and share the common vertex V1.

Another example: it is required to go around the vertices of a face F1 in order, starting with V1

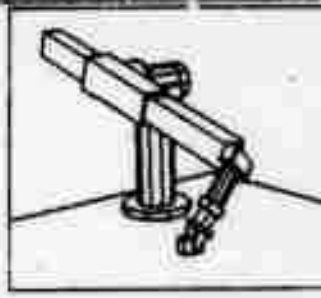
```
T ← V1;
FOREACH E,H | EDGE ⊛ F1 ≡ E
∧     END  ⊛ E  ≡ T
∧     END  ⊛ E  ≡ H
∧     H ≠ T DO BEGIN
        IF H = V1 THEN DONE;
        <statement>;
        T ← H END;
```

When an instance is identified; a new item is created which is associated with the prototype as follows:

```
INSTANCE ⊛ CUBE ≡ INST1                                    [Eq. 3.7]
```

The position and orientation of the instance are expressed as a transformation matrix which relates prototype coordinates to instance coordinates (see Subsection A.3). This 4 x 4 transformation is stored as the datum of the instance.

Most calculations can be performed by transforming the instance back to the prototype rather than by transforming the prototype out to the instance.



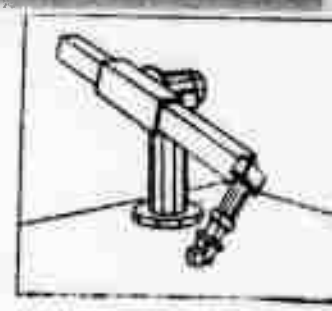
Consider for example the problem of finding the support face of a body by finding through which face the weight vector passes. We could transform each face out to the instance, calculate if the weight vector was in the general direction of the outward pointing normal and then determine whether the weight vector actually passes through the face. This would require that we transform each vertex of the face to the instance. It is more efficient to transform the weight vector back to the prototype by using the inverse transform and check through which face it passes, avoiding all the other transformations, which take of the order of 0.5 m sec. each.

With this "prototype-instance" scheme we can represent all plane faced objects. There is sufficient information available for the vision program to be able to identify objects in two dimensional scenes. In the next section we will show that there is also sufficient information for the arm program.

3.2 ORIENTATION VECTORS

The prototype description is used when it is required to move an instance of some prototype. Apart from the positional information, which is obtainable directly from the instance transform, the prototype description is used to calculate how the instance may be picked up. Knowing how a body is oriented and where it is located does not specify a hand position which may be used to pick the instance up. Although there are an infinite number of ways in which an object may be picked up, we will limit the possibilities by the following heuristics. We will require that the object be picked up by two parallel faces on an axis containing the center of mass, as this will prevent the object from rotating. One but not both surfaces may be replaced by an apex of the body. Both surfaces may be replaced by edges if a normal from the edge intersects the center of mass. These heuristics define a set of orientation vectors. If the hand is positioned at the center of mass with one of these orientation vectors, it will be in a position to grasp the object (see Figure 3.3).

To find systematically all the possible orientation vectors the program first makes a list of all the vectors from the center of mass of the object that 1) intersect and are normal to any edge, 2) intersect and are normal to any face, 3) pass through any apex. Such vectors are known as contact vectors. This list is then searched for pairs of anti-parallel vectors, being careful not to take both vectors from the third class. This is done in the following manner: To find the contact vectors for faces the program simply checks that the perpendicular from the center of mass to the plane lies inside the face and thus the surface is perpendicular at the contact point, or:



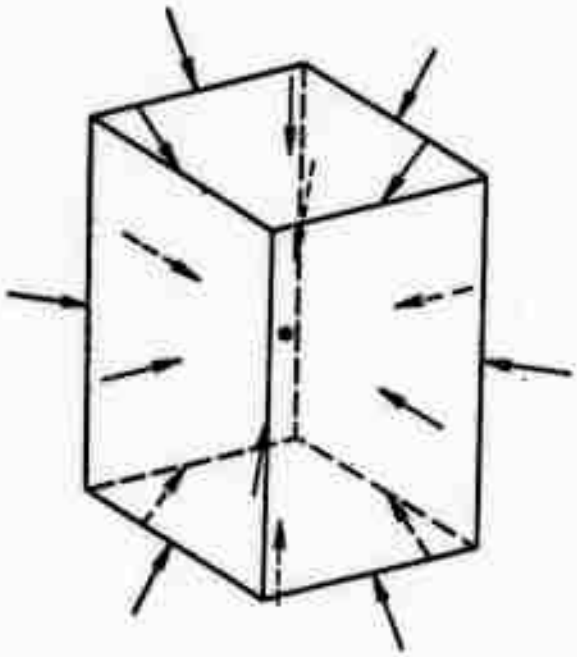
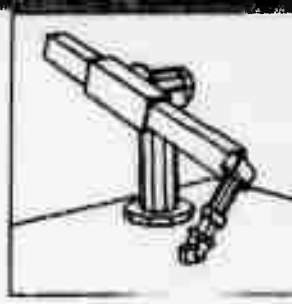


Figure 3.3
Orientation Vectors

```
        FOREACH F | FACE⊙PROTOTYPE≡F DO
        BEGIN
                D←DATUM(F)[4];
                IF D>0.0 THEN GO TO NFACE;
        Comment if this plane is used the center of mass
        will not lie between the finger tips;
                IF CONTAINED(DATUM(F),F) THEN FOUND ONE;
        Comment if the point of intersection of the
        normal and the face within the boundary then
        put the point in the list of contact vectors;
NFACE:  END;
```

The procedure CONTAINED counts the number of region boundary crossings of a ray from the point to infinity. If the number is odd then the point is inside the region, if even then it is outside.

To find the contact vectors for edges it is necessary that a perpendicular from the center of mass intersect the edge. It is also required that the edge represent an outside corner. By the following algorithm the program satisfies the first condition:



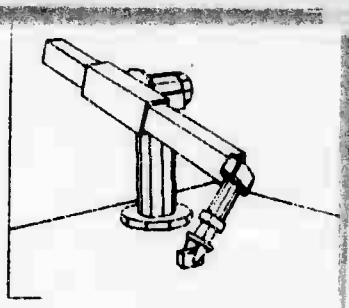
```
FOREACH E,A,B|EDGE⊛PROTOTYPE≡E
        ∧       END⊛E≡A
        ∧       END⊛E≡B
        ∧        A≠B    DO BEGIN
        T←A . (A - B)/(A - B) . (A - B);
        Comment T is the directed distance from
        end point A to the normal, divided by the
        directed distance of A from B;
        IF T≤0.0 ∨ T≥1.0 THEN GO TO NEDGE;
        Comment the normal intersects on the edge;
        C ← A - ((A - B) * T);
        Comment C is a vector from the center of
        mass and perpendicular to the edge at the
        point of contact (see Figure 3.4).

        Now check that this is an outside edge;
        FOREACH N1,N2|BOUNDARY⊛N1≡E
                 ∧    BOUNDARY⊛N2≡E
                 ∧    N1≠N1 DO;
                N1 ← DATUM(N1);
                N2 ← DATUM(N2);
                N ← N1 X N2;
        comment N is a reference vector such that
        we move outside as we rotate about N from
        N1 to N2;
                V ← N1 X C;
                IF N . V ≤ 0
        Comment C points to the outside of
        the vertex;
                THEN FOUND ONE;
        END;
END;
```

To determine contact vectors at verticies we can use only the outside corners of the object (see Figure 3.5). That is, for all edges at this vertex the angle θ must be less than 90 degrees (see Figure 3.5).

```
FOREACH V | VERTEX ⊛ BDY ≡ V DO BEGIN
        C2←V . V;
        FOREACH E,A | END ⊛ E ≡ V
                ∧     END ⊛ E ≡ A
                ∧     A ≠ V  DO
        IF A . V > C2 THEN GO TO NOGOOD;
        FOUND ONE;
NOGOOD: END;
```

The program then searches this list of contact points looking for pairs of anti-parallel vectors, being careful not to take both vectors from the class of vertices. These then are the orientation vectors which are stored with the prototype together with the contact information:



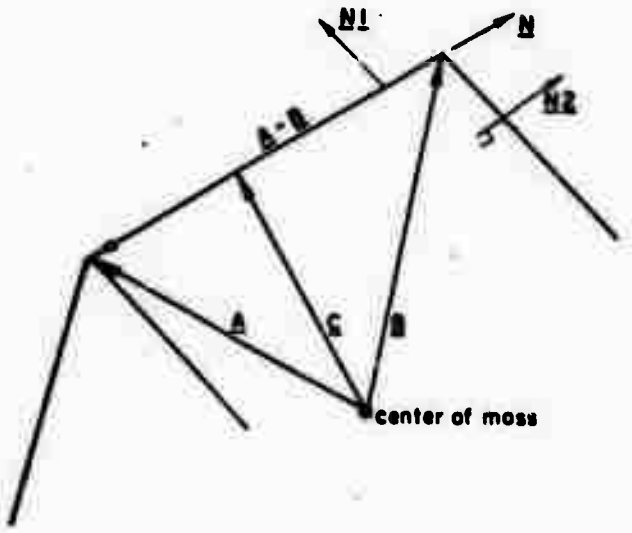


Figure 3.4
Pick-up Point on an Edge

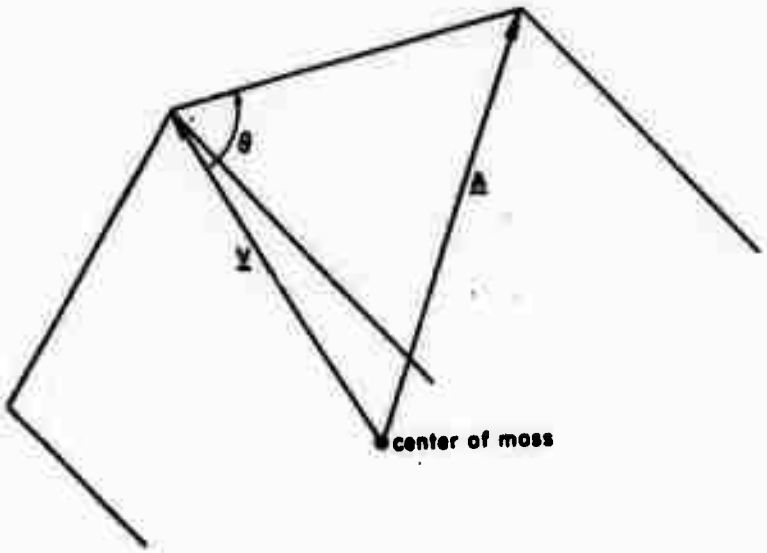
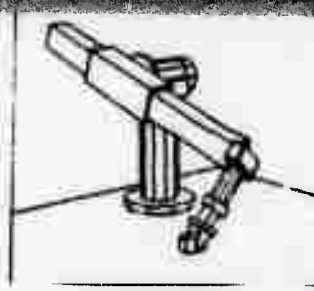


Figure 3.5
Outside Vertex

ORIENTATIONS



ORIENTATION ⊗ BDY ≡ O1
ORIENTATION ⊗ BDY ≡ O2
" " "
ORIENTATION ⊗ BDY ≡ O9

[Eq. 3.8]

CONTACT ⊗ O1 ≡ F1
CONTACT ⊗ O1 ≡ F3
CONTACT ⊗ O2 ≡ F2
CONTACT ⊗ O2 ≡ F4
" " "
CONTACT ⊗ O9 ≡ E6
CONTACT ⊗ O9 ≡ E2

[Eq. 3.9]

The contact information is used in determining which orientation vectors can be used for a given instance. The datum of an orientation vector is a 5 element matrix with the following elements: if $\underline{C1}$ and $\underline{C2}$ are two anti-parallel contact vectors then the datum of the orientation vector is:

$$O[1] = C1[1] \,/\, \left(C1[1]\uparrow 2 + C1[2]\uparrow 2 + C1[3]\uparrow 2 \right)^{(1/2)}$$

$$O[2] = C1[2] \,/\, \left(C1[1]\uparrow 2 + C1[2]\uparrow 2 + C1[3]\uparrow 2 \right)^{(1/2)}$$

$$O[3] = C1[3] \,/\, \left(C1[1]\uparrow 2 + C1[2]\uparrow 2 + C1[3]\uparrow 2 \right)^{(1/2)}$$

$$O[4] = | \underline{C1} |$$

$$O[5] = | \underline{C2} |$$

In the case of a rectangular parallelepiped of size 1.25in. × 1.25in. × 2.55in. the program computes the following 9 orientation vectors:

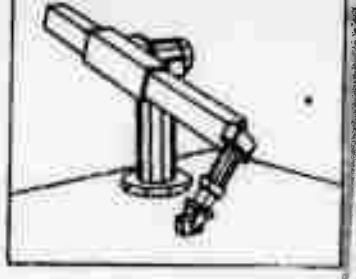
Table 3.1

Orientation Vectors for Rectangular Parallelepiped

O[1]	O[2]	O[3]	O[4]	O[5]
1.00	0.00	0.00	1.60	-1.60
0.00	1.00	0.00	1.60	-1.60
0.00	0.00	1.00	0.78	-0.78
0.71	0.71	0.00	1.13	-1.13
0.71	-0.71	0.00	1.13	-1.13
0.44	0.00	0.90	0.70	-0.70
0.44	0.00	-0.90	0.70	-0.70
0.00	0.44	-0.90	0.70	-0.70
0.00	0.44	0.90	0.70	-0.70

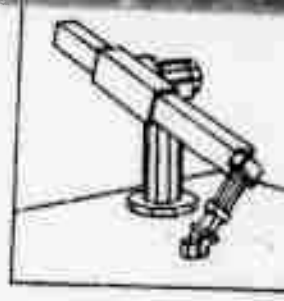
Each row in Table 3.1 represents an orientation vector. The first three elements O[1], O[2] and O[3] give the direction components of the vector. The two contact vectors are given by:

ORIENTATIONS



$$\underline{C1} = \begin{vmatrix} O[1] \\ O[2] \\ O[3] \\ O[4] \end{vmatrix} \qquad \underline{C2} = \begin{vmatrix} O[1] \\ O[2] \\ O[3] \\ O[5] \end{vmatrix}$$

All arm operations with bodies can be reduced to manipulations of these vectors. To pick up a body we need only consider the orientation vectors. We select an orientation vector and specify an approach angle; we can then complete the hand transformation (see Subsection 2.1) and obtain an arm solution.



SECTION 4

MOVE INSTANCE

The procedure MOVE_INSTANCE is called when it is desired to have the arm move an object. MOVE_INSTANCE is first required to find a hand position which enables the hand to grasp the object located by | Ti |. It should then find a second hand position from which the hand can release the object such that it will be specified by | Tf |. The suffixes "i" and "f" refer to the initial and final positions.

4.1 RANGE OF SOLUTION

In manipulating objects the program needs to find arm configurations at both the initial and final positions. To do this the program finds the range of possible approach angles for each orientatior vector; it then chooses some approach angle from the intersection of the two ranges. Given a position and an orientation vector, the program needs to compute the range of approach angle throughout which the arm can reach the object subject to the physical constraints of the arm (see Figure 4.1).

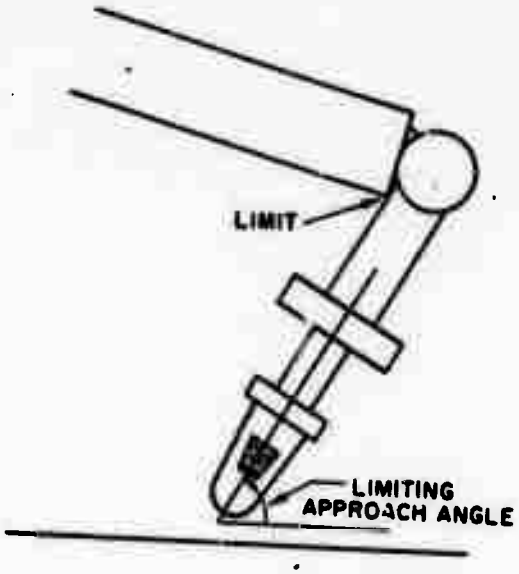
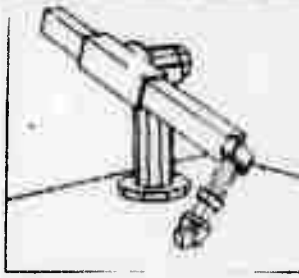


Figure 4.1
Approach limited by Arm

Heuristics are used to find an approach angle at which an arm solution will exist, if one exists at all, and a further test is made to determine if



solutions exist for all approach angles. If a solution exists, but not for the full range, the limits of the range are found by conducting a binary search, using the arm solution procedure to test for feasibility.

In order to ensure that the arm does not penetrate the support plane (see Figure 4.2), the program computes the range of approach angle for which link 5 is above the plane.

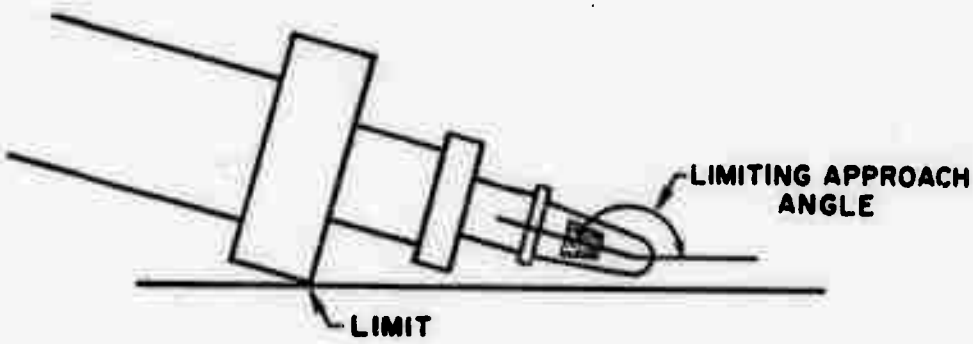


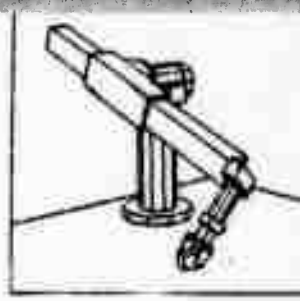
Figure 4.2
Approach Angle Limited by Support Plane

This range of approach angle is then intersected with the possible range of approach angle defined by the arm to give a range throughout which the arm can reach the object limited by both the arm and the support.

This process is repeated for the support post of the arm. A final intersection of ranges is made to keep the hand from intersecting the post, and the limiting range of approach angle is obtained. Possible conflicts with adjacent objects are not considered.

4.2 MOVE INSTANCE

The first action of MOVE_INSTANCE is to select those orientation vectors of the prototype (see section 3.2) which can be used to grasp the object. To do this the program transforms a gravity vector back to the prototype, using the inverse transform, and determines through which face it passes. We consider this to be



the support face. Any orientation vector (Equation 3.8) which has a contact point (Equation 3.9) on this face or on any edge or vertex of this face is discarded; the remaining orientation vectors are marked as possible. This procedure is performed for the object in its initial position and again in its final position. If the action is to be accomplished in one move then the set of available orientation vectors that can be used is the intersection of the two sets of orientation vectors for the initial and final positions. As the most stable way to pick up an object is by grasping the object by its faces, the orientation vectors are ordered by length and the shortest, representing face-face contacts, are considered first.

For each of the orientation vectors so ordered the range of approach is calculated at both the initial and final positions. In order to relate these two ranges for a given orientation vector we make use of the reference approach vector Equation 2.7. By transforming the reference approach vector RAi at the initial position to the final position by:

$$RAi' \leftarrow | Tf | * | Ti |^{-1} * | RAi | \qquad \text{[Eq. 4.1]}$$

the shift S, between approach ranges may then be calculated as:

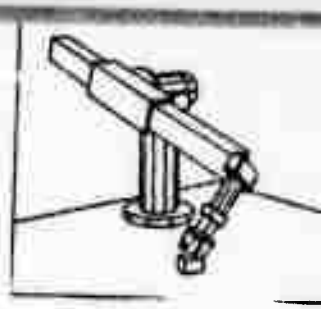
$$S \leftarrow \text{the angle between } RAi' \text{ and } RAf \text{ about } Of \qquad \text{[Eq. 4.2]}$$

After applying the shift S to the initial range the two ranges of approach are intersected. If the intersection is not empty then the solution has been found. An approach vector A is picked within the range of intersection. In order to keep the arm clear of other objects, the preferred approach direction is straight down. Two arm solutions are obtained, one at the initial position and one at the final position, such that the shift between approach vectors is maintained.

If there is no intersection between the first two ranges, the next orientation vector in the intersection of the available set of orientation vectors is tried.

When the set of orientation vectors is empty at either the initial or final position, there is no way that the move can be accomplished as the arm cannot reach the object. If the intersection of the set of orientation vectors were empty, or if after evaluating the ranges for all the orientation vectors the intersection of the ranges was empty then an intermediate position is tried.

In the case of the rectangular parallelepiped whose orientation vectors we obtained (Table 3.1) we will consider the problem of moving an instance from:



1.00	.00	.00	20.00
.00	1.00	.00	30.00
.00	.00	1.00	1.30
.00	.00	.00	1.00

to:

-1.00	.00	.00	40.00
.00	1.00	.00	20.00
.00	.00	-1.00	1.30
.00	.00	.00	1.00

This move includes turning the object upside down.

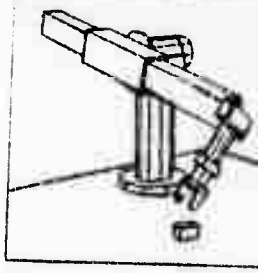
For each of the possible orientation vectors the range of approach is calculated at both positions. P1 is the position vector and O1 is the orientation vector at the initial position and at the second position the vectors are P2 and O2.

P1 = 20.00 30.00 1.30 1.00
O1 = .00 1.00 .00 1.00
Range From 40 to 173 degrees
Shifted Range From 220 to 353 degrees

P2 = 40.00 20.00 1.30 1.00
O2 = .00 1.00 .00 1.00
Range From 7 to 120 degrees
Common range 0 degrees

P1 = 20.00 30.00 1.30 1.00
O1 = 1.00 .00 .00 1.00
Range From 62 to 173 degrees
Shifted Range From 242 to 353 degrees

P2 = 40.00 20.00 1.30 1.00
O2 = -1.00 .00 .00 1.00
Range From 7 to 123 degrees
Common range 0 degrees



P1 = 20.00 30.00 1.30 1.00
Q1 = .71 -.71 .00 1.00
Range From 49 to 173 degrees
Shifted Range From 229 to 353 degrees

P2 = 40.00 20.00 1.30 1.00
Q2 = -.71 -.71 .00 1.00
Range From 29 to 165 degrees
Common range 0 degrees

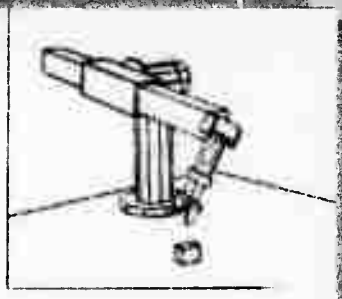
P1 = 20.00 30.00 1.30 1.00
Q1 = .71 .71 .00 1.00
Range From 59 to 173 degrees
Shifted Range From 239 to 353 degrees

P2 = 40.00 20.00 1.30 1.00
Q2 = -.71 .71 .00 1.00
Range From 7 to 115 degrees
Common range 0 degrees

As the intersection of all the ranges is zero, an attempt is now made to make a plan to move the object from its initial position to an intermediate position and then from the intermediate position to the final position. The calling program specifies whether an intermediate position is to be tried and if so where. A clear space in front of the arm will give the greatest range of approach in terms of joint motion constraints; however, choosing the initial or final position as the intermediate position ensures that the space is clear in which to place the object. Even when the initial position is used as the intermediate position, the move is usually accomplished, as it is only the position and not the orientation that is specified. On most occasions when the arm is unable to carry out a task in one move, it is because, as in the present example, the object must be turned over. It is then necessary for the arm to turn the object part-way over, put it down, and pick it up again to complete the move.

Given the new intermediate position, a set of possible new support faces is determined as the intersection of the set of faces which are neighbors of the original support face and of the final support face. A transformation is constructed for this intermediate position to bring the new support face parallel to the original support surface. To give maximum freedom of approach range, the object is then turned about an axis normal to the support to make the object's principal axes perpendicular to the shoulder of the arm. If the distance from the center of mass to the original support face is different from the initial distance, the height of the center of the object s adjusted.

The process is then repeated to find a common approach. First a move solution to



the intermediate position is computed and then, if successful, a move solution from the intermediate position to the final position is attempted . Ranges are saved during the "initial - final" move attempt as they are needed in the "initial - intermediate," "intermediate - final" move evaluations.

In the example we are considering the intermediate position is specified as 30, 30, 1.3 and as the intersection of all the ranges was zero an intermediate position is set up as:

.00	.19	-.98	30.00
.00	.98	.19	30.00
1.00	.00	.00	.65
.00	.00	.00	1.00

The program now tries to make a plan to move the object to this position:

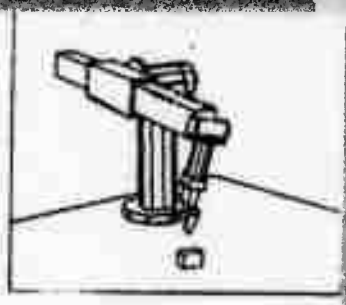
P1 = 20.00 30.00 1.30 1.00
Q1 = .00 1.00 .00 1.00
Range From 40 to 173 degrees
Shifted Range From -50 to 83 degrees

P2 = 30.00 30.00 .65 1.00
Q2 = .19 .98 .00 1.00
Range From 19 to 156 degrees
Common range From 19 to 83 degrees
Approach = 51 degrees

A common range exists and the program now calculates the two hand positions:

-.63	.00	-.78	20.00
.00	1.00	.00	30.00
.78	.00	-.63	1.30
.00	.00	.00	1.00

-.76	.19	.62	30.00
.15	.98	-.12	30.00
-.63	.00	-.78	0.65
.00	.00	.00	1.00



The program now tries to make a plan to move the object from the intermediate position to the final position:

```
P1 =  30.00  30.00    .65   1.00
Q1 =    .19    .98    .00   1.00
  Range   From 19 to 156 degrees
Shifted Range From -71 to 66 degrees

P2 =  40.00  20.00   1.30   1.00
Q2 =    .00   1.00    .00   1.00
  Range   From  7 to 120 degrees
Common range From  7 to 66 degrees
      Approach = 36 degrees
```

Once again a common range exists and the program calculates the two hand positions:

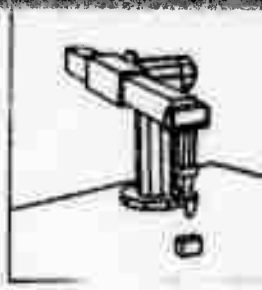
```
|  -.79    .19   -.58  30.00 |
|                             |
|   .15    .98    .11  30.00 |
|                             |
|   .59    .00   -.81    .65 |
|                             |
|   .00    .00    .00   1.00 |

|  -.59    .00    .81  40.00 |
|                             |
|   .00   1.00    .00  20.00 |
|                             |
|  -.81    .00   -.59   1.30 |
|                             |
|   .00    .00    .00   1.00 |
```

The problem is solved and the moves can be made.

By this procedure it is possible to make any re-positionings and re-orientations, even when the goal must be accomplished in two moves.

About 4 seconds are required to compute the arm positions when two moves must be used.



SECTION 5

TRAJECTORIES

5.1 GENERAL CONSIDERATIONS

In moving the arm we have two positions, the initial and final. The discussion until now has emphasized these positions and their determination (Subsection 4.2). In this section we will describe the move in detail.

The simplest solution is to move the joints independently from their initial position to their final position, using a simple servo. Consider the situation shown in Figure 5.1, where the hand is turning a block onto its side. The motion is mostly in joint 5; if all the joints were moved to their final positions then the hand would try to move through the support. What is needed is to lift the arm up and down as joint 5 is moved, in order to clear the support. When the arm starts to move, it is normally working with respect to some surface, for instance, picking up a block from a table. As it starts to move the motion of the hand should be directly away from the surface. If we were to specify a position on a normal to the surface out from the initial position, and then to require that the hand pass through this position, we would achieve the correct departure motion. If we could further specify the time required to reach this position, we could control the speed at which the block was to be lifted.

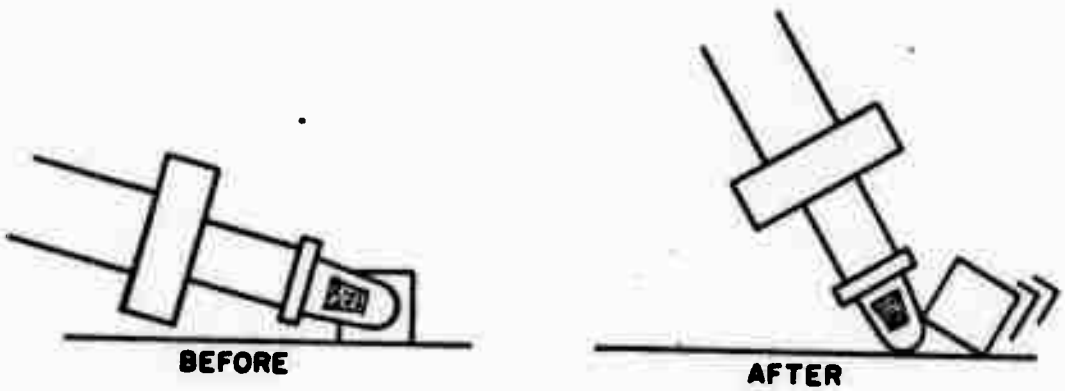
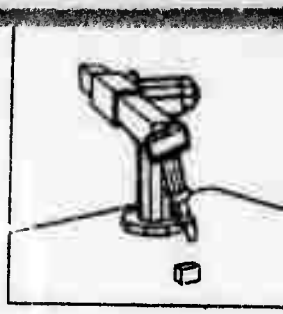


Figure 5.1
Crash



To estimate how far this position should be from the surface, consider Figure 5.2, as this represents the worst case of surface penetration. If the hand had been lifted by the maximum surface penetration, approximately 25% the length of the last link, the collision would have been avoided.

For such an initial move, the differential change of joint angles is calculated (Subsection 2.3) for a move of 3 inches in the direction of the outward pointing normal. A time to reach this position based on a low arm force is then calculated. The same set of requirements exists in the case of the final position. Here we wish once again to approach the surface in the direction of the normal, this time passing down through a letdown point.

We now have 4 positions: initial, liftoff, letdown, and final and if we were to servo the arm from one position to the next we would not collide with the support (see Figure 5.3).

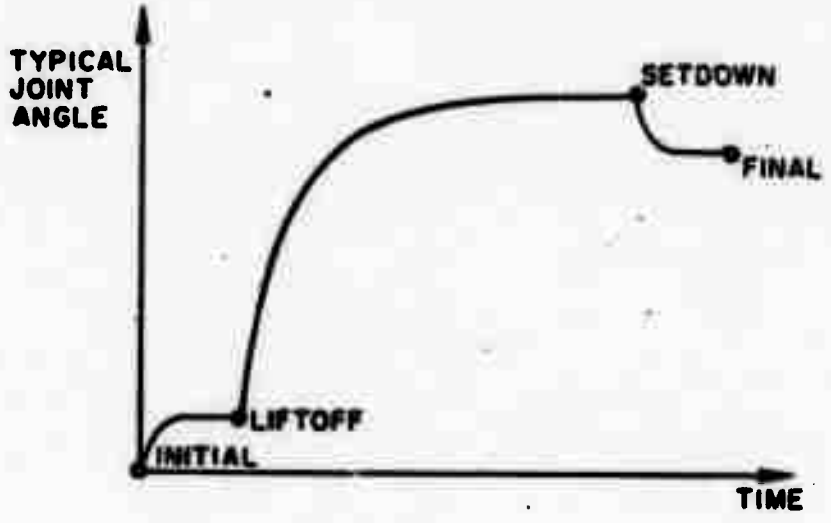
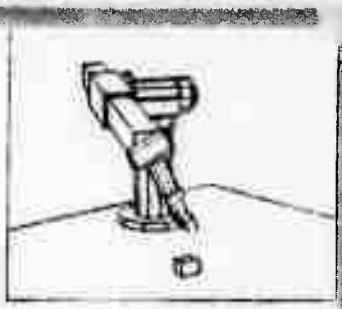


Figure 5.3
Point to Point Trajectory

We would, however, like the arm to start and end its motion with zero velocity and acceleration. Further, there is no need to stop the arm at all the intermediate positions. We require only that the joints of the arm pass through the trajectory points corresponding to these intermediate positions at the same time.

The time for the arm to move through each trajectory segment is calculated as follows: for the initial and final segments the time is based on the rate of approach of the hand to the surface and is some fixed constant. The time



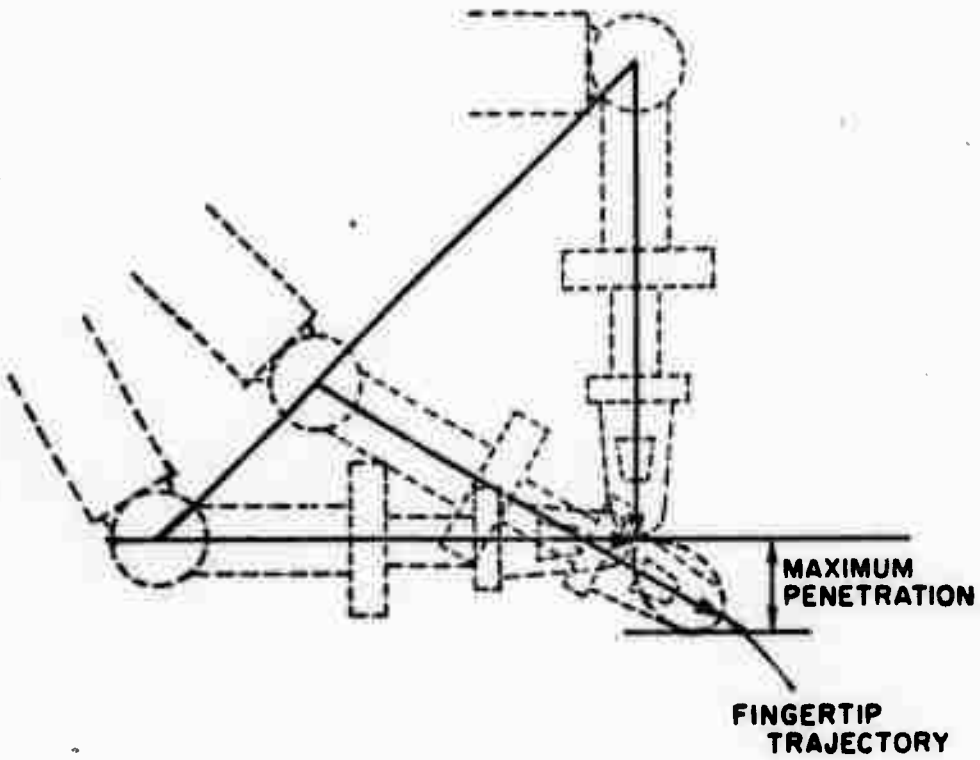
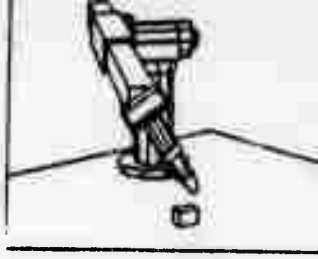


Figure 5.2
Maximum Penetration of Hand



necessary for each joint to move through its mid trajectory segment is estimated, based on a maximum joint velocity and acceleration. The maximum of these times is then used for all the joints to move through the mid trajectory segment.

Knowing the joint variables and times we can determine a polynomial for each joint, expressing joint angle as a function of time, which passes through all the points and has zero initial and final velocity and acceleration; as there are 4 points and 4 velocity and acceleration constraints we would need a 7th. order polynomial. Although such polynomials satisfy our conditions, they often have extrema between the initial and final points and the joint variable must be evaluated at each extremum to check that it has not exceeded the working range of the joint.

As the extrema are difficult to evaluate for such high order polynomials, we use a different approach. We specify three polynomials for each joint, one for the trajectory from the initial point to the liftoff point, a second from the liftoff to the setdown point, and a third from the setdown to the final point. We specify that velocity and acceleration should be zero at the initial and final points and that they should be continuous at the intermediate points. This sequence of polynomials satisfies our conditions for a trajectory and has extrema which are easily evaluated.

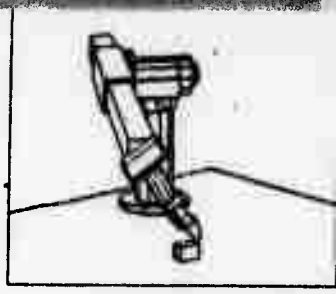
If a joint exceeds its working range at an extremum, then the trajectory segment in which it occurs is split in two, a new intermediate point equal to the joint range limit is specified at the break, and the trajectory recalculated (See Figure 5.4).

Although a collision avoider has not been implemented, except in the case of the table and the arm support post, such a program would modify the arm trajectory in the same manner by specifying additional intermediate points. If a potential collision were detected one or more jointe would be required to pass through some additional trajectory points in order to avoid the collision.

We have another type of trajectory that we wish to be able to compute, one which moves the arm along a well defined space curve. Here we obtain a sequence of joint angles at points along the space curve. The velocity along the space curve is controlled by relating distance along the curve to time between points (See primitive DRAW Subsection 7.2). This type of curve leads to a trajectory with many points. If we were to use a single polynomial it would need to be of high order, for this reason the sequence of low order polynomials is also preferred.

5.2 POLYNOMIALS

For each trajectory segment we have position, velocity and acceleration constraints at each end. Except at the beginning and end of the trajectory the velocity and acceleration constraints are continuity constraints. There are



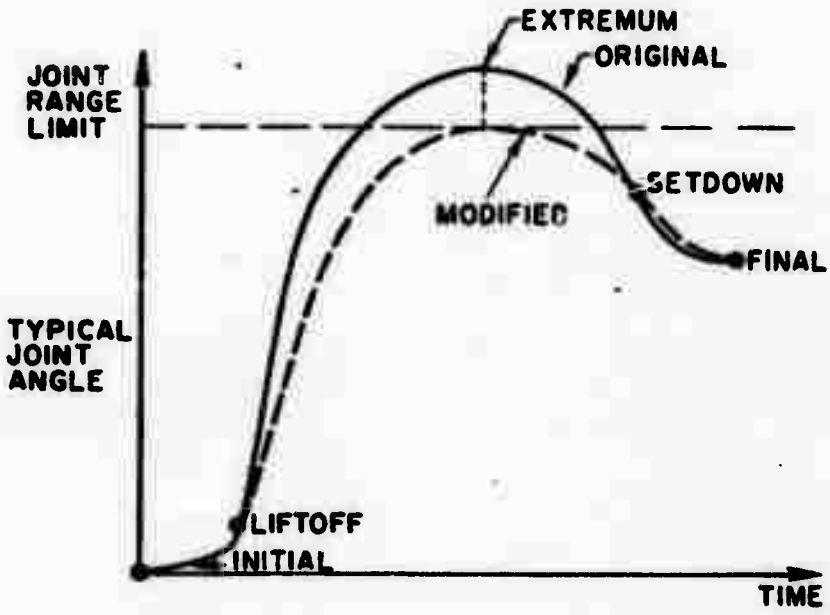


Figure 5.4
Trajectory beyond Joint Range

only four constraints on the intermediate trajectory segments and five constraints at the ends. Thus for the first and last trajectory segments a fourth order polynomial will suffice and for the intermediate trajectory segments a third order polynomial will be needed.

Consider a trajectory segment described by:

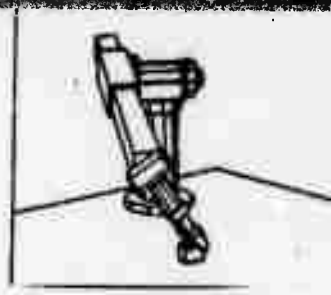
$$\theta = Ai4*t'^{4} + Ai3*t'^{3} + Ai2*t'^{2} + Ai1*t' + Ai0 \qquad \text{[Eq. 5.1]}$$

with

$$t' = t/\tau i \qquad \text{[Eq. 5.2]}$$

where Aij is the coefficient of the jth. power of the ith. trajectory segment, and time t, is normalized to unity at the end of the trajectory segment of duration τi.

For the first trajectory segment at time t'=0:



$$\theta 0 = A10 \quad \text{[Eq. 5.3]}$$

$$\dot{\theta} 0 = 0 + A11 \quad \text{[Eq. 5.4]}$$

$$\ddot{\theta} 0 = 0 + A12 \quad \text{[Eq. 5.5]}$$

and at time t'=1:

$$\theta 1 - \theta 0 = \Delta\theta 1 = A14 + A13 \quad \text{[Eq. 5.6]}$$

$$\tau 1 * \dot{\theta} 1 = 4*A14 + 3*A13 \quad \text{[Eq. 5.7]}$$

$$\tau 1^2 * \ddot{\theta} 1 = 12*A14 + 6*A13 \quad \text{[Eq. 5.8]}$$

for the last trajectory segment we substitute:

$$t'' = t' - 1 \quad \text{[Eq. 5.9]}$$

and at time t''=0:

$$\theta n = An0 \quad \text{[Eq. 5.10]}$$

$$\dot{\theta} n = 0 + An1 \quad \text{[Eq. 5.11]}$$

$$\ddot{\theta} n = 0 + An2 \quad \text{[Eq. 5.12]}$$

and at time t''=-1:

$$\theta n - \theta(n-1) = \Delta\theta n = -An4 + An3 \quad \text{[Eq. 5.13]}$$

$$\tau n * \dot{\theta}(n-1) = -4*An4 + 3*An3 \quad \text{[Eq. 5.14]}$$

$$\tau n^2 * \ddot{\theta}(n-1) = 12*An4 - 6*An3 \quad \text{[Eq. 5.15]}$$

For the general ith. trajectory segment we have

$$0 = Ai4 \quad \text{[Eq. 5.16]}$$

$$\theta(i-1) = Ai0 \quad \text{[Eq. 5.17]}$$

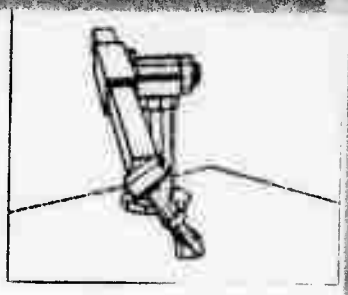
$$\tau i * \dot{\theta}(i-1) = Ai1 \quad \text{[Eq. 5.18]}$$

$$\tau i^2 * \ddot{\theta} i(-1) = Ai2 \quad \text{[Eq. 5.19]}$$

$$\theta i - \theta(i-1) = \Delta\theta i = Ai3 + Ai2 + Ai1 \quad \text{[Eq. 5.20]}$$

$$\tau i * \dot{\theta} i = 3*Ai3 + 2*Ai2 + Ai1 \quad \text{[Eq. 5.21]}$$

$$\tau i^2 * \ddot{\theta} i = 6*Ai3 + 2*Ai1 \quad \text{[Eq. 5.22]}$$



Equations 5.3, 5.4, 5.5, 5.10, 5.11, 5.12, and Equations 5.17 specify coefficients directly. The remaining coefficients may be solved in the form:

$$\begin{bmatrix} \Delta\theta 1 \\ 0 \\ 0 \\ \Delta\theta 2 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \Delta\theta n \end{bmatrix} = \left[\begin{array}{cc|ccc|c} 1 & 1 & & & & \\ 3/\tau 1 & 4/\tau 1 & -1/\tau 2 & & & \\ 6/\tau 1^2 & 12/\tau 1^2 & 0 & -2/\tau 2^2 & & \\ & & 1 & 1 & 1 & \\ & & 1/\tau 2 & 2/\tau 2 & 3/\tau 2 & \\ & & & 2/\tau 2^2 & 6/\tau 2^2 & \\ & & & & & \\ \hline & & & & & -3/\tau n \quad 4/\tau n \\ & & & & & 6/\tau n^2 \quad -12/\tau n^2 \\ & & & & & 1 \quad -1 \end{array}\right] * \begin{bmatrix} A13 \\ A14 \\ A21 \\ A22 \\ A23 \\ \vdots \\ An3 \\ An4 \end{bmatrix}$$

[Eq. 5.23]

where the blocks indicated in Equation 5.23 may be repeated for each additional point that the trajectory must pass through.

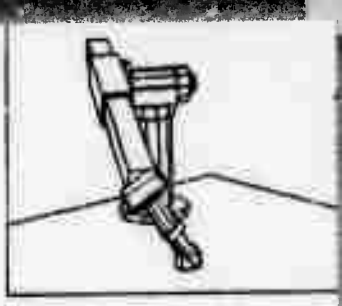
Normalized time t' runs from 0 to 1 for each trajectory segment except for the last segment in which case normalized time t'' runs from -1 to 0. The arm servo program requires that normalized time t' run from 0 to 1 for all trajectory segments. If we substitute:

$$t' = t'' + 1 \qquad \text{[Eq. 5.24]}$$

in:

$$\theta = A4n*t''^4 + A3n*t''^3 + A2n*t''^2 + A1n*t'' + A0n \qquad \text{[Eq. 5.25]}$$

we obtain:



$$\theta = A4n*t'^4 + (-4*A4n + A3n)*t'^3 + (6*A4n - 3*A3n + A2n)*t'^2 + (-4*A4n + 3*A3n - 2*A2n + A1n)*t' + (A4n - A3n + A2n - A1n + A0n)$$

[Eq. 5.26]

and this gives us the coefficients of a polynomial for the last trajectory segment in which normalized time runs from 0 to 1 as required for the servo program.

5.3 TRAJECTORY EXTREMA

Intermediate trajectory segments are described by third order polynomials, one such polynomial for each joint:

$$\theta = Ai3*t'^3 + Ai2*t'^2 + Ai1*t' + Ai0 \qquad \text{[Eq. 5.27]}$$

the derivative is:

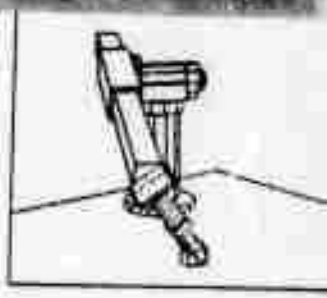
$$\dot{\theta} = 3*Ai3*t'^2 + 2*Ai2*t' + Ai1 \qquad \text{[Eq. 5.28]}$$

The times of the extrema are given as the roots of Equation 5.28:

$$t' = -1/3\ Ai2/Ai3 \pm \{(1/3\ Ai2/Ai3)^2 - 1/3\ Ai1/Ai3\}^{1/2} \qquad \text{[Eq. 5.29]}$$

If the discriminant is positive and $0 < t' < 1$ then an extremum exists and it can be evaluated by Equation 5.27.

In the case of the initial and final trajectory segments the trajectories are described by fourth order polynomials (Equation 5.1) with the low order terms missing (see Equation 5.4, 5.5, 5.11, and 5.12).



$$\theta = Ai4*t'^4 + Ai3*t'^3 + Ai0 \qquad \text{[Eq. 5.30]}$$

and the derivative by:

$$\dot{\theta} = 4*Ai4*t'^3 + 3*Ai3*t'^2 \qquad \text{[Eq. 5.31]}$$

The time of the extremum is given as the roots of Equation 5.31:

$$t' = -3/4*Ai3/Ai4 \qquad \text{[Eq. 5.32]}$$

if $0 < t' < 1$ then an extremum exists and the value of θ is:

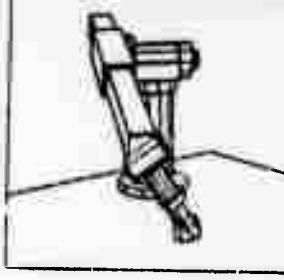
$$\theta = -1/4*Ai3*t'^3 \qquad \text{[Eq. 5.33]}$$

In the case of the first and last segments we also require monotonic motion, in order to avoid overshoot. To ensure monotonicity we compare the value at the extremum to the initial or final point instead of to the joint physical limit.

5.4 CONTINUOUS MOTION

In Subsection 5.1 we mentioned trajectories on space curves, where arm solutions are obtained at regular intervals along the curve and a trajectory is then planned through these positions. There is a special case of such trajectories in which the first and last arm positions are the same, for example the circular arm motion of Figure 5.5. The position of the hand is described by an angle; the orientation remains the same during the motion. Solutions are obtained every 60 degrees so that the hand will approximately follow the path of the circle. In the case of the beginning and end, two additional positions are specified, "b" and "h," at 20 and 340 degrees respectively. The arm is given the same period of time for each of the 8 segments, and thus the arm will accelerate from rest, move around the circle and come back to rest. The first and last small segments ensure smooth departure and arrival.

For continuous motion we provide an alternate path from position "g" through position "a" to position "c" (see Figure 5.5) requiring continuity of both velocity and acceleration at these points. The hand is caused to move in a circular path continuously having first been started from point "a" by causing it to move along the alternate trajectory at point "g." When it is desired to stop it is not diverted at the "switch" at position "g" but allowed to stop by moving through position "h" to "a."



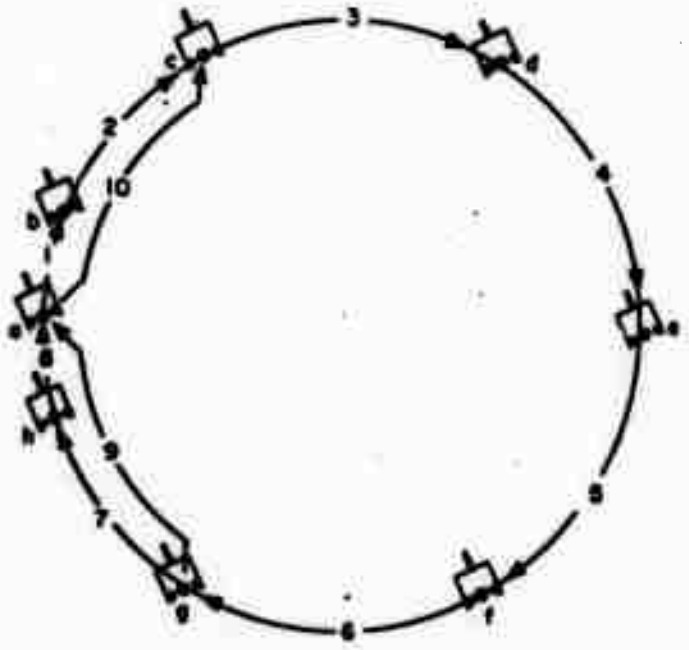
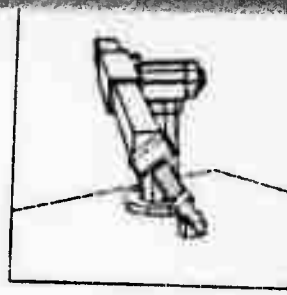


Figure 5.5
Looping

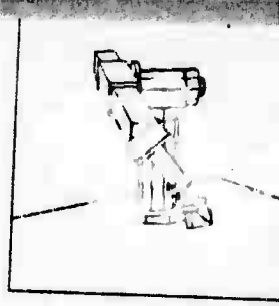
In the following matrix equation the x's represent non zero elements of Equation 5.23. Equation 5.34 is for the case of a simple trajectory and Equation 5.35 is in the case of looping, providing for the solution of two additional fourth order trajectory segments 9 and 10.



$$
\begin{vmatrix} \Delta\theta_1 \\ 0 \\ 0 \\ \Delta\theta_2 \\ 0 \\ 0 \\ \Delta\theta_3 \\ 0 \\ 0 \\ \Delta\theta_4 \\ 0 \\ 0 \\ \Delta\theta_5 \\ 0 \\ 0 \\ \Delta\theta_6 \\ 0 \\ 0 \\ \Delta\theta_7 \\ 0 \\ 0 \\ \Delta\theta_8 \end{vmatrix} = \begin{vmatrix} XX & & & & & & & \\ XXX & & & & & & & \\ XX\ X & & & & & & & \\ & XXX & & & & & & \\ & XXXX & & & & & & \\ & XX\ X & & & & & & \\ & & XXX & & & & & \\ & & XXXX & & & & & \\ & & XX\ X & & & & & \\ & & & XXX & & & & \\ & & & XXXX & & & & \\ & & & XX\ X & & & & \\ & & & & XXX & & & \\ & & & & XXXX & & & \\ & & & & XX\ X & & & \\ & & & & & XXX & & \\ & & & & & XXXXX & & \\ & & & & & XXXX & & \\ & & & & & XX & & \end{vmatrix} * \begin{vmatrix} A13 \\ A14 \\ A21 \\ A22 \\ A23 \\ A31 \\ A32 \\ A33 \\ A41 \\ A42 \\ A43 \\ A51 \\ A52 \\ A53 \\ A61 \\ A62 \\ A63 \\ A71 \\ A72 \\ A73 \\ A83 \\ A84 \end{vmatrix}
$$

[Eq. 5.34]

In the case of looping, the blocks of elements marked as Y's and Z's are the same.



$$
\begin{vmatrix} \Delta\theta1 \\ 0 \\ 0 \\ \Delta\theta2 \\ 0 \\ 0 \\ \Delta\theta3 \\ 0 \\ 0 \\ \Delta\theta4 \\ 0 \\ 0 \\ \Delta\theta5 \\ 0 \\ 0 \\ \Delta\theta6 \\ 0 \\ 0 \\ \Delta\theta7 \\ 0 \\ 0 \\ \Delta\theta8 \\ 0 \\ 0 \\ \Delta\theta9 \\ 0 \\ 0 \\ \Delta\theta10 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} XX & & & & & & & & \\ XXX & & & & & & & & \\ XX\ X & & & & & & & & \\ & XXX & & & & & & & \\ & XXXY & & & & & & & \\ & XX\ Y & & & & & & & \\ & & XXY & & & & & & \\ & & XXXX & & & & & & \\ & & XX\ X & & & & & & \\ & & & XXX & & & & & \\ & & & XXXX & & & & & \\ & & & XX\ X & & & & & \\ & & & & XXX & & & & \\ & & & & XXXX & & & & \\ & & & & XX\ X & & & & \\ & & & & & XXX & & & \\ & & & & & ZZZX & & & \\ & & & & & ZZ\ X & & & \\ & & & & & & XXX & & \\ & & & & & & XXXXX & & \\ & & & & & & XXXX & & \\ & & & & & & & XX & \\ & & & & & ZZZ & & X & \\ & & & & & ZZ & & X & \\ & & & & & & & XXXX & \\ & & & & & & & XXXXX & \\ & & & & & & & XXX\ X & \\ & & & & & & & & XXXX \\ & Y & & & & & & & XXXX \\ & Y & & & & & & & XXX \end{vmatrix} * \begin{vmatrix} A13 \\ A14 \\ A21 \\ A22 \\ A23 \\ A31 \\ A32 \\ A33 \\ A41 \\ A42 \\ A43 \\ A51 \\ A52 \\ A53 \\ A61 \\ A62 \\ A63 \\ A71 \\ A72 \\ A73 \\ A83 \\ A84 \\ A91 \\ A92 \\ A93 \\ A94 \\ A101 \\ A102 \\ A103 \\ A104 \end{vmatrix}
$$

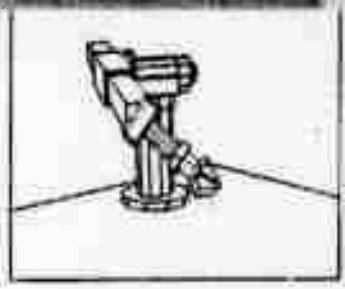
[Eq. 5.35]

Equation 5.35 is the same as Equation 5.34 except for the addition of the last eight equations for segments 9 and 10. These are fourth order polynomials due to the extra end continuity constraints. The groups of non zero elements in the lower triangle of Equation 5.35 relate to the continuity requirements at points g and c (see Figure 5.5).

5.5 DIFFERENTIAL MOTION

The arm can also make differential motions where all six joints are required to change a certain amount in a given time. In this case all the joints are driven together to make the change in the specified time. The changes of joint angles are determined as in Subsection 2.3.

Joint angle θ as a function $g(t')$ of normalized time t', for a change of $\Delta\theta$ is:

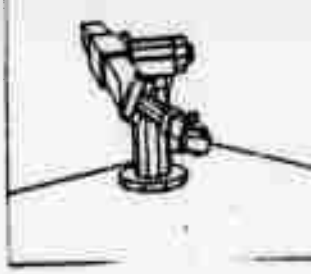


$$t' = t/\tau m \qquad \text{[Eq. 5.36]}$$

$$\theta = g(t') = \Delta\theta * t'^{3}(6t'^{2} - 15t' + 10) \qquad \text{[Eq. 5.37]}$$

This gives zero initial and final acceleration and velocity.

This type of motion has all the undesirable properties described in Subsection 5.1 and is only suitable when small changes are to be made, as in correcting the hand position during visual servoing. Such motions are of course much simpler to plan than regular trajectory controlled motion.



SECTION 6

SERVO

In this section we relate position error to joint torque in the arm servo. We then discuss the model of the joint drive in order to convert joint torque into motor drive. In the final section provision is made for degrees of freedom of the arm as it is servoed.

6.1 FEEDBACK LOOP

In this section we will describe the servo response. We will treat the system as continuous, and will ignore the effects of sampling, assuming that the sampling period is much less than the response time of the arm. At the end of the section we will check that this assumption is valid. Time is normalized to the sampling period, which has the effect of scaling the link inertia up by f↑2 where f is the sampling frequency. The Laplace transform is used throughout and is represented by F(s).

The set point for each joint of the arm is obtained by evaluating the appropriate trajectory segment polynomial for the required time. The velocity and acceleration are evaluated as the first and second derivatives of the polynomials.

The position error is the observed position θ less the required value θs. Likewise the velocity error is the observed velocity less the required velocity. Position feedback is applied to decrease position error and velocity feedback is used to provide damping.

The feedback loop is shown in Figure 6.1 The arm is represented by $1/s^2 J$, where J is the effective link inertia a function of arm configuration. T(s) is an external disturbing torque. The set point R(s) is subtracted from the current position to obtain the position error E(s) and is multiplied by s, representing differentiation, to obtain the error velocity. There are two feedback gains ke and kv, position and velocity respectively.

By writing the loop equation we can obtain the system response:

$$E(s) = (-s^2 J)/(s^2 J + skv + ke)*R(s) + 1/(s^2 J + skv + ke)*T(s) \qquad \text{[Eq. 6.1]}$$

and the condition for critical damping is:

$$kv = 2(J*ke)^{1/2} \qquad \text{[Eq. 6.2]}$$

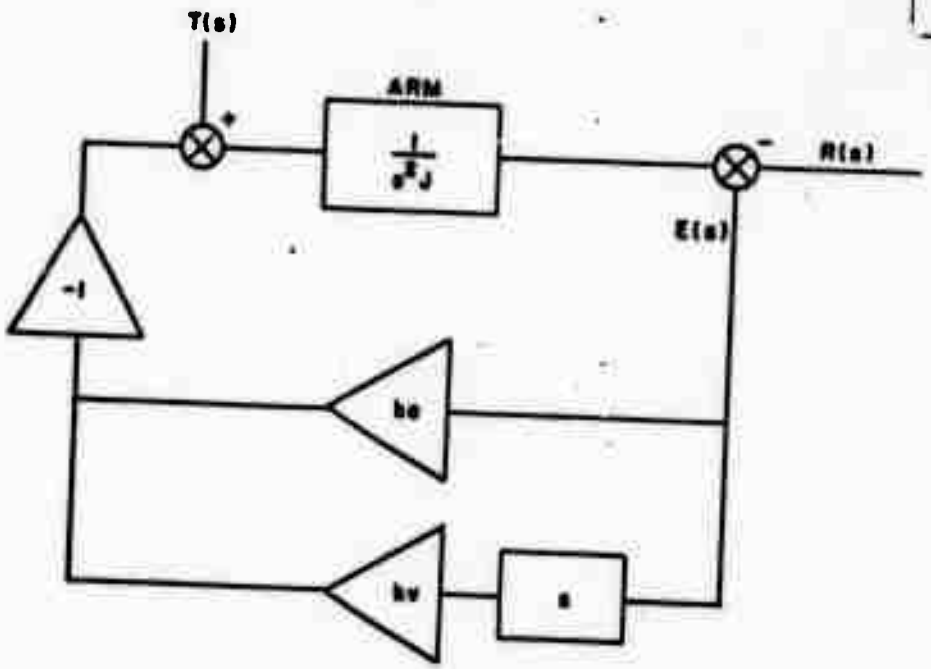


Figure 6.1
Simple Servo Loop

It can be seen that the system response is dependent on J as would be expected. Because the effective link inertia J can vary by 10:1 as the arm configuration changes, we are unable to maintain a given response (see Equation 6.2) independent of arm configuration. If however we add a gain of -J as shown in Figure 6.2 then we obtain:

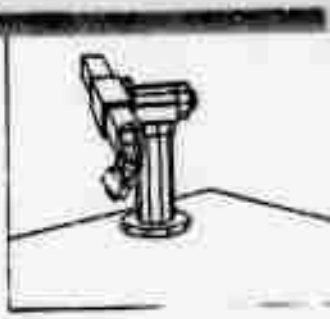
$$E(s) = (-s^2)/(s^2 + skv + ke)*R(s) + 1/(s^2 + skv + ke)*T(s)/J \qquad \text{[Eq. 6.3]}$$

and the condition for critical damping is:

$$kv = 2*(ke)^{1/2} \qquad \text{[Eq. 6.4]}$$

It can be seen that the servo response is now independent of arm configuration.

The principal disturbing torque is that due to gravity, causing a large position error, especially in the case of joint 2. If we were able to add a term equal to the negative of the gravity loading Tg (see Figure 6.3) then we would obtain the same system response as in Equation 6.3 except that T would become Te, the external disturbing torque, less the gravity dependent torque, reducing the position error.



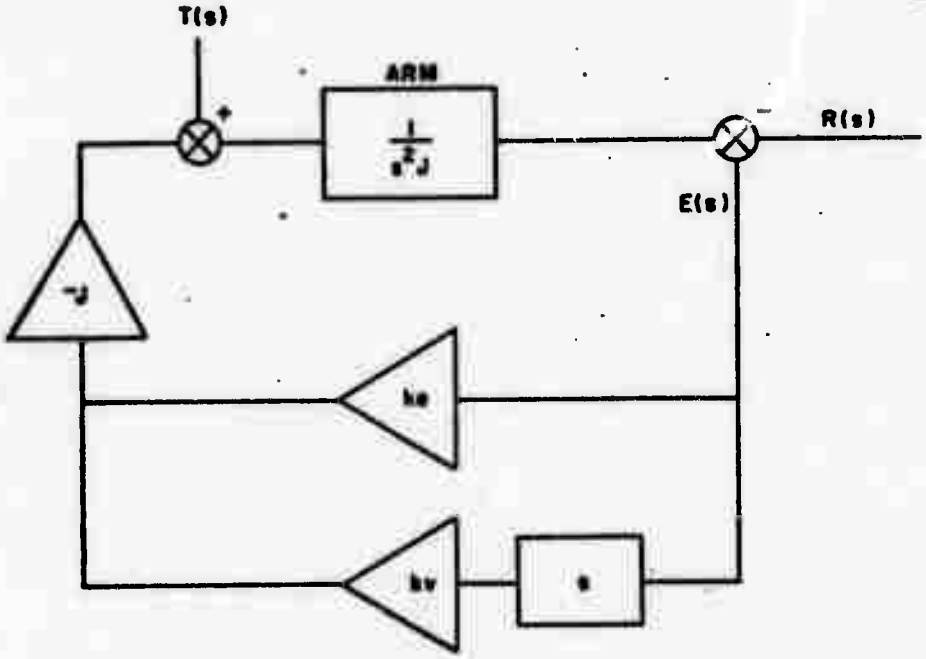


Figure 6.2
Effective Inertia Independent Feedback

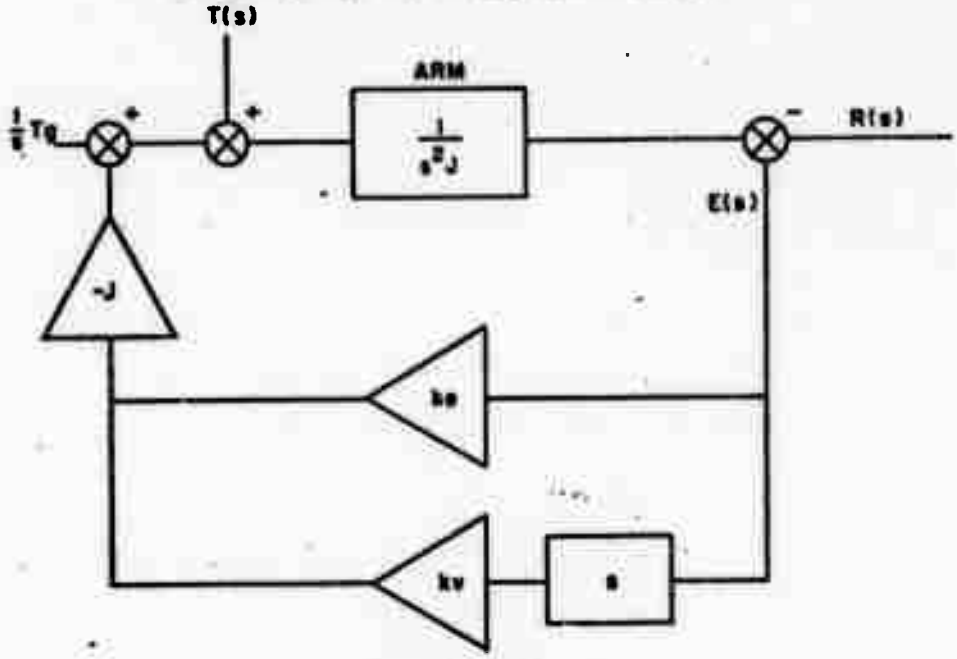
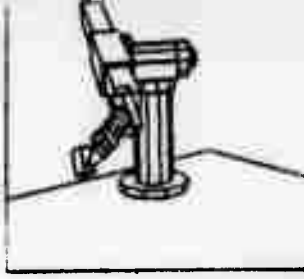


Figure 6.3
Gravity Independent Loop



$$E(s) = (-s^2)/(s^2 + skv + ke)*R(s) + 1/(s^2 + ekv + ke)*Te(e)/J \qquad \text{[Eq. 6.5]}$$

We can compensate for the effect of acceleration of the set point R(s), the first term in Equation 6.5, if we add a term $s^2 R(s)$ (see Figure 6.4) and obtain a system response:

$$E(s) = 1/(s^2 + ekv + ke)*T(e)/J \qquad \text{[Eq. 6.6]}$$

The gain of -J and the torque Tg are obtained from Equation 2.36 which we restate here:

$$Fi = \sum_{j=i}^{n} \sum_{k=1}^{j} \text{Trace}(|Ujk|*|Hj|*|Uji|^T * \ddot{q}k)$$

$$+\sum_{j=i}^{n} \sum_{k=1}^{j} \sum_{p=1}^{j} \text{Trace}(|Ujkp|*|Hj|*|Uji|^T * \dot{q}j * \dot{q}k)$$

$$-\sum_{j=i}^{n} mj*|G|*|Uji|*\bar{R}j \qquad \text{[Eq 2.36]}$$

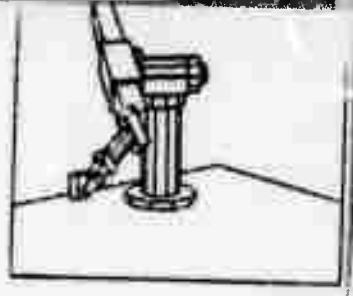
What is desired is to obtain an expression of the form:

$$Fi = Ji * \ddot{q}i + Tg \qquad \text{[Eq. 6.7]}$$

given an arm configuration qi.

The velocity dependent terms of Equation 2.36 are only significant at high speed and are small for the arm we are using. We will ignore the second term of Equation 2.36, although it could be included with the third term as the values of $\dot{q}j$ and $\dot{q}k$ are known from the trajectory.

We may interchange the order of s[illegible]tion of Equation 2.36 to obtain:



$$Fi = \sum_{k=1}^{6}\sum_{j=i}^{6} \mathrm{Trace}(|Ujk|*|Hj|*|Uji|^{T})*\ddot{q}k$$

$$-\sum_{j=i}^{6} mj*|G|*|Uji|*\bar{R}j \qquad \text{[Eq. 6.8]}$$

or:

$$Fi = \sum_{k=1}^{6} Cik\,\ddot{q}k + Ci \qquad \text{[Eq. 6.9]}$$

where:

$$Cik = \sum_{j=i}^{6} \mathrm{Trace}(|Ujk|*|Hj|*|Uji|^{T}) \qquad \text{[Eq. 6.10]}$$

and:

$$Ci = -\sum_{j=i}^{6} mj*|G|*|Uji|*\bar{R}j \qquad \text{[Eq. 6.11]}$$

One further simplification is that we may disregard the terms of Equation 6.9 in which i≠k as they are very small, to obtain:

$$Fi = Ji * \ddot{q}i + Tg \qquad \text{[Eq. 6.12]}$$

where:

$$Ji = \sum_{j=i}^{6} \mathrm{Trace}(|Uji|*|Hj|*|Uji|^{T}) \qquad \text{[Eq. 6.13]}$$

and Tg as Ci in Equation 6.11

Ji is the effective joint inertia and Tg is the constant term due to gravity.

The servo has uniform system response under varying arm configurations and is compensated for gravity loading and for the acceleration of the set point r.

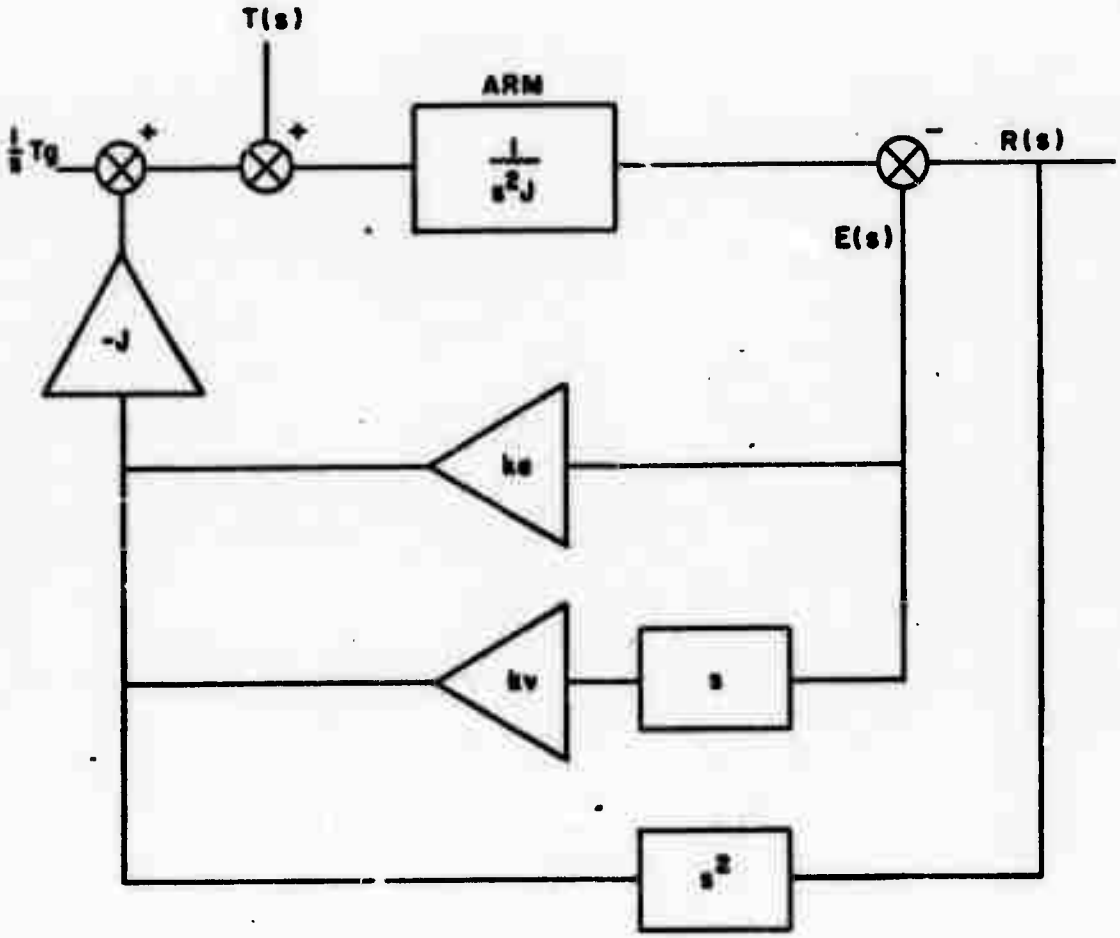
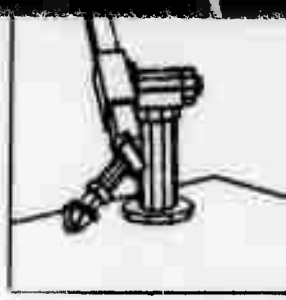


Figure 6.4
Acceleration Compensated Loop



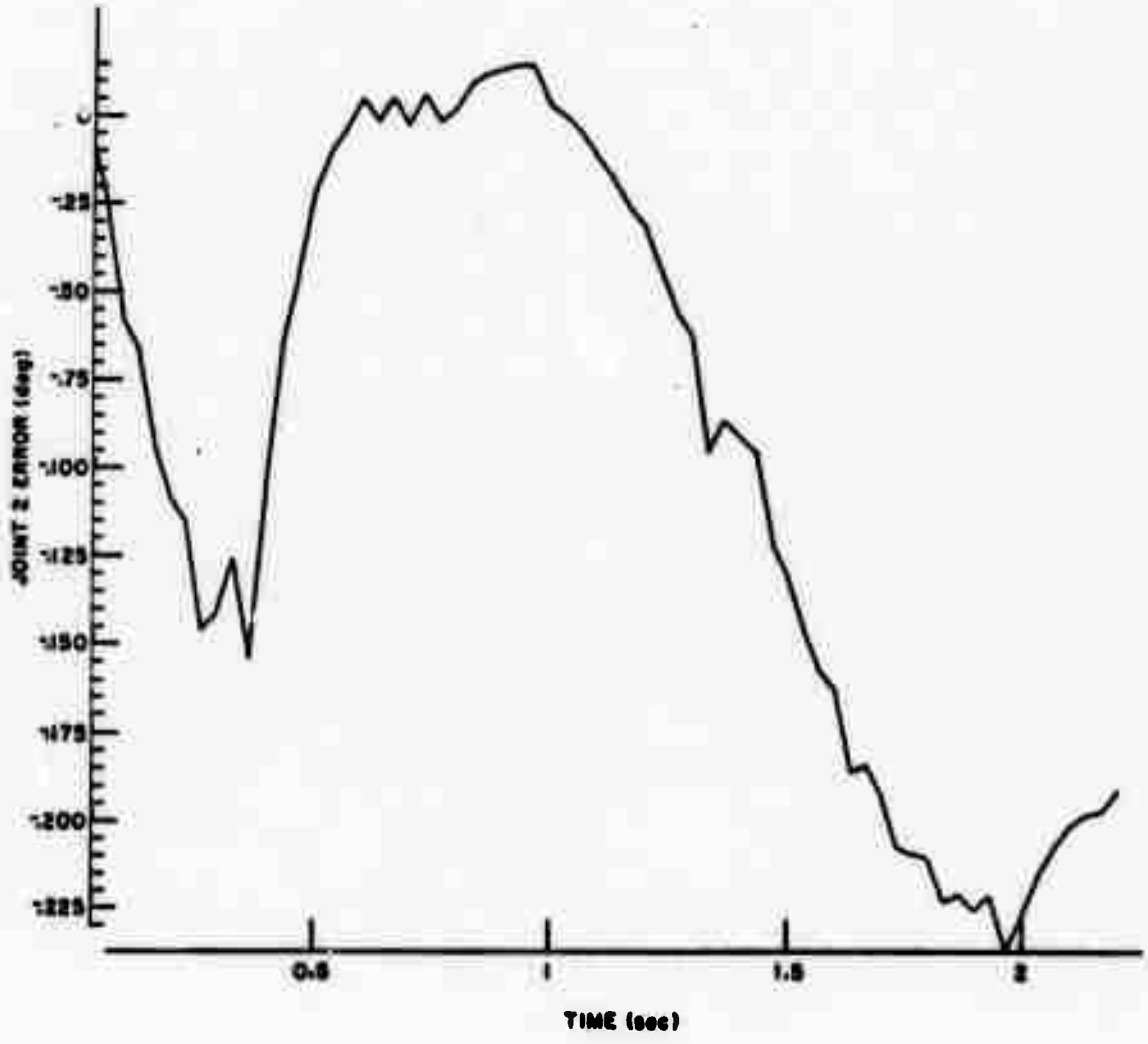
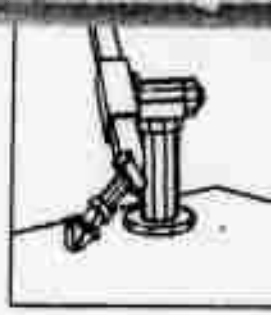


Figure 6.5
Position Error with Inertia Compensation



Three plots of position error v. time for joint 2 are shown: Figure 6.5, Figure 6.6, and Figure 6.7.

In Figure 6.5 there is no gravity or feedback compensation, in Figure 6.6 the gravity term is added, and in Figure 6.7 the acceleration compensation is included.

The steady state error for a given error torque Te is:

$$Ess = Te/(J * ke) \qquad \text{[Eq. 6.14]}$$

We would like the arm to be stiff when it is servoing and a value of 50 oz/in at the hand is the minimum acceptable. if a lesser value is used the arm moves very sloppily. For this value of stiffness the values of ke can be calculated by means of Equation 6.14. Representative values of kv based on critical damping may then be calculated (Equation 6.4) and will give us the lower limits on ke. and kv.

There is, however, some noise in position measurement and if we are to obtain velocity by differencing observed position readings then the same noise is in the velocity determination. The primary source of noise is the quantizing noise of the analog to digital converter used to determine position. The A/D converter is a 12 bit device and thus has a relative error of 1/4096. The equivalent noise torque at the joint is given by:

$$Tn = k * J * e \qquad \text{[Eq. 6.15]}$$

where k is either ke if the velocity is determined separately, or ke+kv if the velocity is determined by differencing, and e is the quantizing noise. A noise torque of approximately F0/2 (see Subsection 6.2) seems to be acceptable and this gives an upper bound on k.

Based on this information we can evaluate Equations 6.14, 6.4 and 6.15 for each joint:

Table 6.1

Ranges of Servo Gains

JOINT	ke	kv	k
1	0.038	0.39	0.18
2	0.032	0.36	0.33
3	0.005	0.14	0.75
4	0.06	0.49	0.49
5	0.06	0.49	1.6
6	0.06	0.49	50.0

The second column is based on Equation 6.14 and the third on Equation 6.4 These

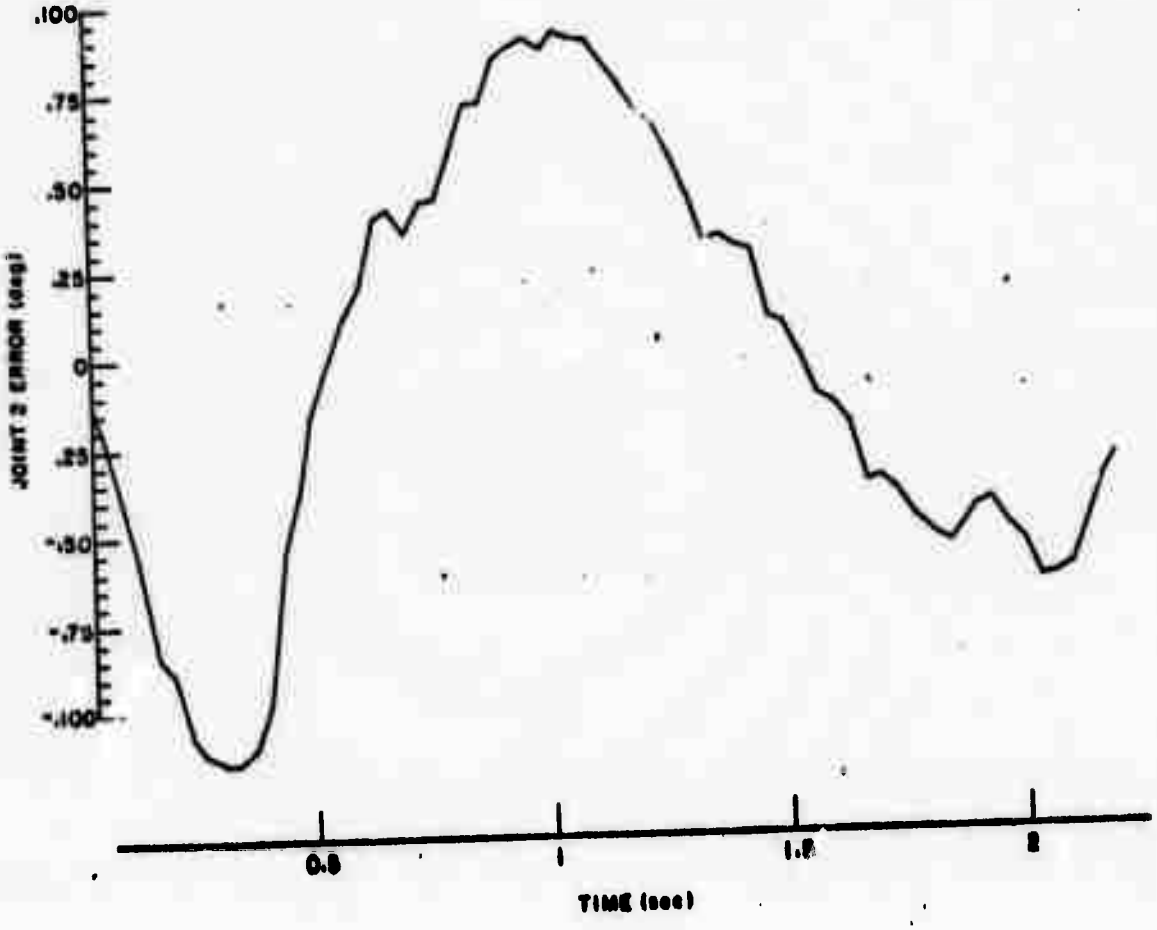
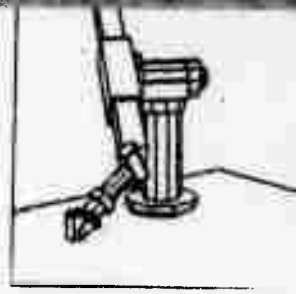


Figure 6.6
Position Error with Gravity Compensation



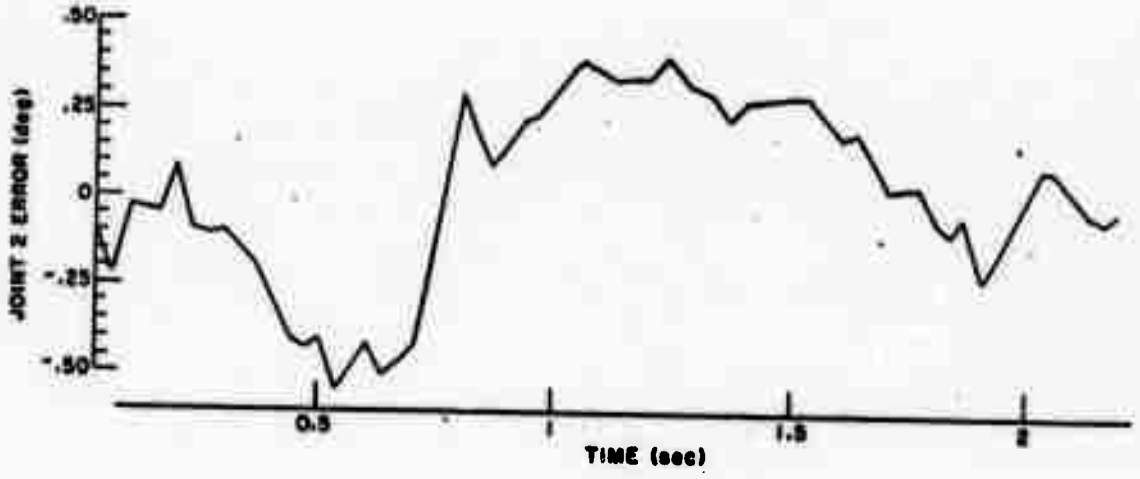
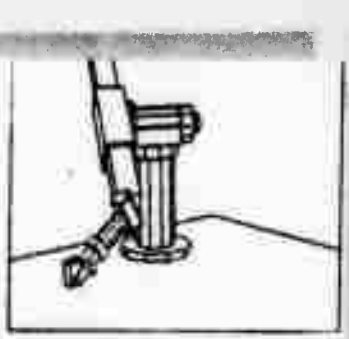


Figure 6.7
Position Error with Acceleration Compensation



are the minimum acceptable gains. The fourth column is from Equation 6.15 and represents the maximum allowable gain if the information is to be obtained by means of position measurement using the available A/D converter. In the case of Joints 1 and 2 it can be seen that an alternative means must be used to obtain velocity information although the position information is acceptable using the A/D converter. To measure the velocity for these two joints a tachometer generator is used, which has much lower noise than velocity obtained by differencing position measurements.

We made an assumption at the beginning of this section that the sampling frequency was much higher than the frequency response of the arm. We can obtain the frequency response from Equation 6.6 by finding the inverse transform. This is found to be:

$$\text{frequency} = kv * f / 2 \qquad \text{[Eq. 6.16]}$$

As maximum kv that we require is 0.49 this give us a response of 1/4 the sampling frequency. It is found in practice that the velocity gain can be increased to 1 before the effects of sampling become apparent.

At the beginning and end of each trajectory and at the end of each trajectory segment, the values of Ji and Tg are evaluated (Equation 6.12). They are then given to the servo together with the part trajectory polynomial coefficients. As the trajectory is executed the values of Ji and Tg are linearly interpolated.

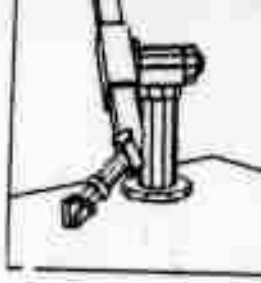
For the hand in the position considered in the example in Subsection 2.1 the values of Ji and Tg are (in the oz. in. 1/60th. second system of units):

Table 6.2

Servo Parameters

JOINT	Ji	Tg
1	761000	0
2	953000	1419
3	9500	-53
4	83000	94
5	82000	37
6	4000	0

Although these gains give an acceptable response from the point of view of stiffness, the gain is too low to maintain the high positional tolerance of ±0.05 in, which we are just able to measure using the 12 bit A/D converter. In order to achieve this error tolerance the position error is integrated when the arm has reached the end of its trajectory. When the position error of a joint is within tolerance the brake for that joint is applied and the joint is no longer servoed. When all the joints are within the error tolerance the trajectory has been executed.



If the arm is to move a heavy load its predicted effects are taken into account by increasing the effective mass and inertia of the last link of the arm before evaluating Equation 6.12. Similarly, if the hand is to exert a given force or moment then the equivalent arm torque (see Subsection 2.5) is added to Tg.

6.2 MOTOR DRIVE

The output of the servo equation is a torque to be applied at the joint. Each joint has an electric motor drive and a harmonic drive gear reduction. The motors are driven by a pulse-width modulated voltage signal. The output of the computer is this pulse-width and the polarity. The drive module relates torque to drive voltage pulse-width.

The motors are driven by a 360 Hertz pulse-width modulated voltage source. The program output "h" is the relative "on" time of this signal. If we plot an experimental curve of "h" v. joint torque we obtain two discontinuous curves depending on the joint velocity (see Figure 6.8).

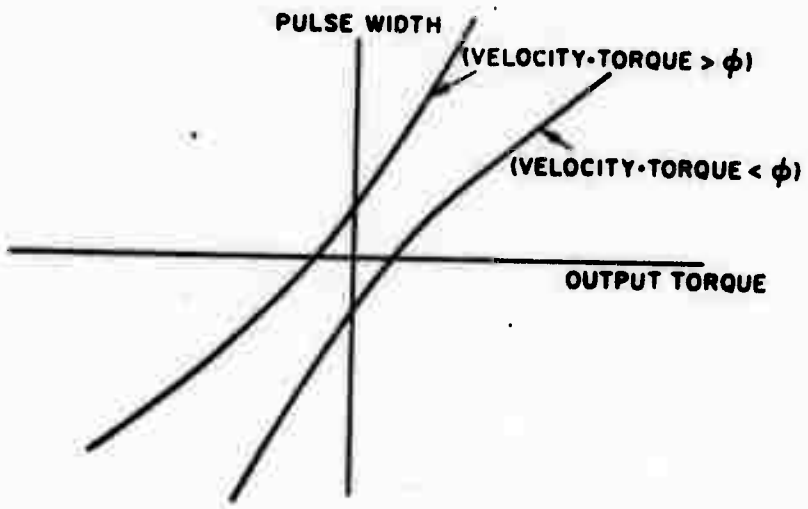
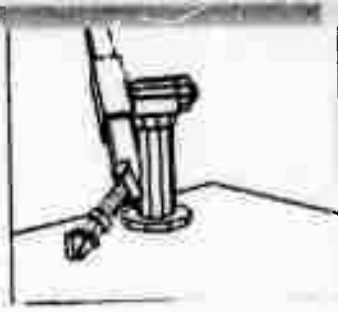


Figure 6.8
Pulse Width v. Output Torque

This curve can be explained in terms of two friction effects: load dependent, causing the two curves to diverge, and load independent, causing separation at the two curves at the origin. The electrical motor time constant also affects the shape of the curve near the origin. Experimentally determined curves are supplied to the servo program in the following piecewise linear form (see Figure 6.9)



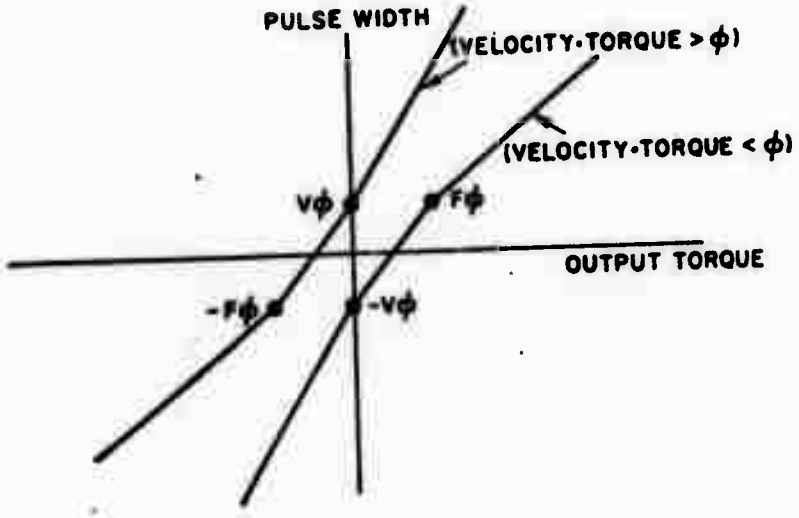


Figure 6.9
Piecewise Linear Pulse Width v. Torque

where:

V0 is the motor drive at which the joint will move at constant velocity exerting zero force in the direction of motion;

F0 is the force that the joint will exert at drive level V0 but with a negative velocity;

the slopes and slope differences are obtained from the experimental curves.

When the velocity is very low the direction of intended motion is substituted for the velocity.

One other factor considered is the back emf of the motor. The value of "h" is the ratio of required voltage to supply voltage. The supply voltage is simply augmented by the computed back emf before "h" is calculated.

When the velocity is non zero the output torque is predictable but at zero velocity and with zero intended motion the error in the output torque can be as much as half the horizontal displacement (F0/2) between the two curves at the origin. The values for this error torque at a typical arm configuration in terms of force at the hand FH, are:

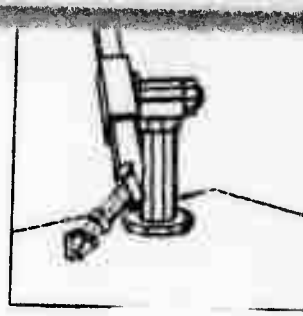


Table 6.3

JOINT	Arm Static Friction F0 oz.	FH oz.
1	400	±10
2	700	±20
3	70	±35
4	100	±5
5	160	±8
6	100	±70

It can be seen that the arm can exert forces with a typical tolerance of ±10 oz.

6.3 PARTIALLY CONSTRAINED MOTION

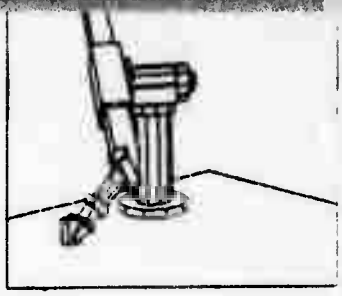
If we require the arm to exhibit a degree of freedom along a given direction or about a given axis at the hand, the program selects a joint to be "free" on the following basis. The program calculates the equivalent arm torque (Subsection 2.5) for a unit force in the given direction or moment about the given axis, and then normalizes the torques by dividing through by their respective F0 (Subsection 6.2). The program then selects that joint which has the largest normalized torque to be "free." This is the joint which is most sensitive to motion in the required direction and it would be the first joint to move if the force were slowly increased from zero in the free direction.

If we require more degrees of freedom we repeat the process, being careful not to select the same joint twice.

To free the joint during motion, the feedback gains ke and kv are set to zero (see Subsection 6.1). This means that the free joint still has acceleration compensation (Equation 6.6) and gravity compensation (Equation 6.5). If the hand is required to exert an external force this is added so that the joint is compensated for all known forces and has no feedback.

The free joint servo response may be obtained from Equation 6.6 as:

$$E(s) = Te/(J * s^2) \qquad \text{[Eq. 6.17]}$$



SECTION 7

CONTROL

In addition to moving, the arm can perform such functions as opening and closing its hand. These functions and motions are called "primitives" and can be put together to make an "arm program." An arm program is assembled to make a trajectory file specifying the primitives together with supporting data, trajectories, effective inertia constants and gravity loading terms.

Two programs exist, one for assembling "arm programs" and the other for executing the resulting trajectory files.

7.1 ARM STATE

The state of the arm is described by the following global variables, which are located in the upper segment and are available to all other programs sharing this segment, such as a strategy program. At the termination of execution of a program the state variables which describe the arm are updated.

ARM_MOTION The name of the program currently being executed. This is a warning flag to other programs that the arm is in motion.

ARM_WAIT The name of the program execution of which is temporarily suspended.

ARM_STATUS The error state of the arm at the end of execution.

ARM_LINK The 4x4 transform which describes the position and orientation of the hand at the end of execution.

GRASP The separation between the finger tips.

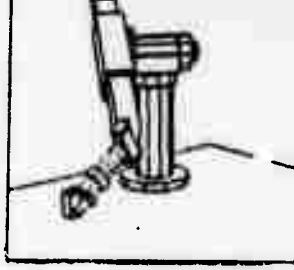
ARM_SEGMENT An integer variable incremented at the beginning and end of each MOVE primitive.

Programs can be executed by the following two procedures.

DO_IT (NAME) causes the program NAME to be executed.

DO_PROCEED causes continued execution of the program execution of which was interrupted.

During execution various errors can occur. Although some errors may be desired states, they are known as errors because they cause the arm to stop and to apply all the brakes. There are various touch sensors on the arm and if, when selected, one of them touches anything, an "error" occurs.



The following is a list of the error messages:

x1 Excessive force at joint x.

2 Hand closed beyond minimum specified opening

x6 Touch sensor x was touched.

22 Excessive force at hand

23 Arm failed to stop on specified force.

There are other messages which refer to system errors, such as file not found.

If an error occurs, execution of the program is suspended and the stat. variables are updated, the contents of ARM_MOTION and ARM_WAIT are exchanged and the error code is set in ARM_STATUS. Execution can proceed with the next primitive, after error recovery, by calling procedure DO_PROCEED.

7.2 PRIMITIVES

This section lists the arm primitives, which have meaning at two times: once at assembly when the trajectory file is being created and feasibility must be checked, trajectories planned etc., and once at execution time when the primitives are executed in the same way that instructions are executed in a computer.

OPEN (DIST) Plan to open or close the hand such that the gap between the finger tips is DIST.

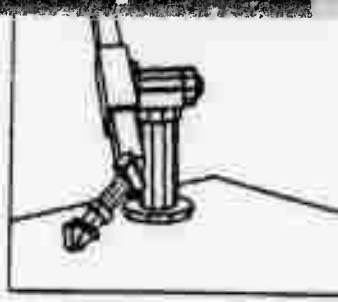
CLOSE (MINIMUM) Plan to close the hand until it stops closing and then check that the gap between the finger tips is greater than MINIMUM. If it is less, then give error 2.

CHANGE (DX_DY_DZ, VELOCITY) Plan to move the arm differentially (Subsection 5.5) to achieve a change of hand position of vector DX_DY_DZ at a maximum speed of VELOCITY.

SWEEP (DIST, VELOCITY) Plan to move the hand differentially (Subsection 5.5) in the direction of the hand's orientation vector a distance DIST at a maximum velocity VELOCITY.

LIFT (DIST, VELOCITY) This is the same as SWEEP except that it is in the direction of $\underline{O} \times \underline{A}$ where O and A are the orientation and approach vectors respectively.

REACH (DIST, VELOCITY) Again this is the same as SWEEP except it is in the direction of the Approach vector.



TILT (ANGLE) Plan to rotate the hand differentially ANGLE degrees (Subsection 5.5) about the orientation vector.

TURN (ANGLE) This is a similar rotation about O × A.

TWIST (ANGLE) Here the rotation is about the approach vector.

PLACE Plan to move the hand vertically down until the hand meets some resistance, that is, the minimum resistance that the arm can reliably detect.

MOVE (T) At assembly time check that the position specified by the hand transformation T is clear. Plan to move the hand along a trajectory from its present position to | T |. The hand is moved up through a point LIFTOFF given by LIFTOFF = INITIAL POSITION + DEPART. where DEPART is a global vector initialized to z = 3 inches. Similarly on arrival the hand is moved down through a point SET DOWN given by: SET DOWN = FINAL POSITION + ARRIVE. ARRIVE is also set to z = 3 inches.

PARK Plan a move as in MOVE but to the "park" position.

DRAW (DX DY DZ, ROT AXIS, θ1, CRANK, CRANK AXIS, θ2, TIME, LOOPS, FORCE, NUMBER_FREE, FREE VECTOR) This is a trajectory motion of the hand. The hand is at the end of a vector (CRANK) which is rotated θ2 degrees around an axis (CRANK AXIS) as its origin is translated (DX DY DZ). At the same time the hand is re-oriented about another axis (ROT AXIS) θ1 degrees. See Figure 7.1. If the end point is the same as the initial then looping may be specified (see Subsection 5.2). Finally a number of degrees of freedom and an excess force to be applied during the program may be specified. With this primitive we can do almost anything!

There are also control primitives which specify how the other primitives are to be carried out.

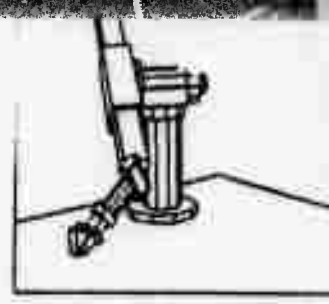
STOP (FORCE, MOMENT) During the next arm motion stop the arm when the feedback force is greater than the equivalent joint force (Subsection 2.5). If the arm fails to stop for this reason before the end of the motion, generate error 23.

SKIPE (ERROR) If error ERROR occurred during the previous primitive then skip the next primitive.

SKIPN (ERROR) if error ERROR occurred during the previous primitive execute the next primitive otherwise skip the next primitive.

JUMP (LAB) Jump to the primitive whose label in LAB.

WAIT Stop execution, update the state variables and wait for a proceed command.



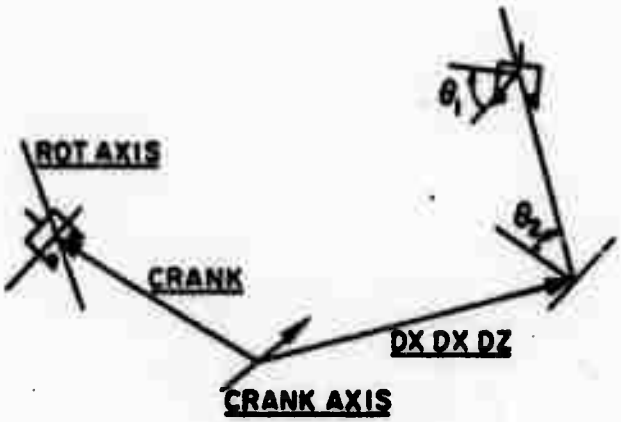


Figure 7.1
A Draw Motion

TOUCH (MASK) Enable the touch sensors specified by mask for the next primitive.

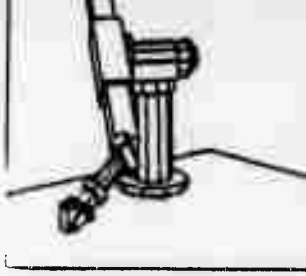
SAVE Save the differential deviation from the trajectory set point. This can be caused by CHANGE type primitives.

RESTORE Cause the arm to deviate from the trajectory set point at the end of the next motion by the deviation last saved.

7.3 ASSEMBLY PROGRAM

The assembly program has two modes of input, message procedure or source file. We will describe the source file input here although it should be kept in mind that a strategy program would be able to execute the same functions.

Planning normally begins from the current position and proceeds from there. The planned state of the arm is kept in a 4x4 transformation STATE. Thus when a series of moves is planned it is always from STATE to the specified transform T that moves are made. At the end of each successful move STATE is updated. In order to start a program a BEGIN pseudo-op must be assembled. This causes a trajectory file to be named and specifies the initial state of STATE. Similarly at the end of a program the trajectory file must be closed. We will list the pseudo-ops:



BEGIN (FILE, T) Open file FILE as the trajectory file and initialize STATE to transformation | T |.

MERGE Merge the last assembled primitive with the preceding motion primitive.

END Close the trajectory file.

MACRO (FILE) Causes input to the planning program to be switched to FILE until the end of FILE. This gives us macros without parameters. The MACROs may be nested.

In the case of most primitives their parameters are either vectors or transformations; should another program use the assembly program these vectors and transformations would be data structures of that program. In the case of source input we need to define such data types and associate them with symbolic names. All names must be defined before a primitive can be assembled.

Data types:

TRANS (NAME, R, X, Y, Z, Ox, Oy, Oz) Set up a 4x4 transformation NAME such that it has position x,y,z and orientation vector Ox,Oy,Oz and that the approach vector is rotated R degrees from the reference approach vector about the orientation vector.

VECT (NAME, x_1, x_2, x_3, x_4) Define a vector NAME of value x_1, x_2, x_3, x_4.

MOVE_INSTANCE (TI, TF, IP) This is partly a data type primitive as it sets up a series of transformations to move an object with transformation TI such that it has transformation TF using intermediate position IP if necessary. However it assembles all the move and hand primitives to accomplish the move.

PROTOTYPE (OBJECT) this sets up the prototype of the body to be moved by MOVE_INSTANCE.

The planning program is 30K and shares a 14K segment which contains runtime routines and global data. Typical running times are 1 sec to plan a move. and from 0.5 to 4 seconds for MOVE_INSTANCE depending on the complexity of the move.

7.4 PROGRAMMING EXAMPLES

We will give some examples of hand programs to clarify the use of primitives. The first example is to move the hand to a position 20,30,1 to pickup an object, then to move it to 40,20,2 and place it on the table.

```
BEGIN TRANSFER 0
TRANS T
```

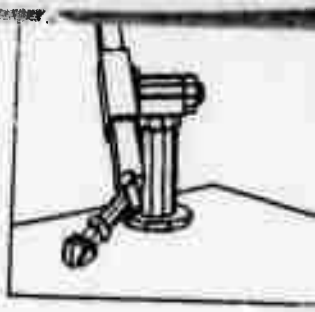
```
        90 20 30 1 1 0 0        Set up a transform to
                                position the hand.
        MOVE T
        CLOSE 0.5               If there is nothing here
                                then an error will occur.
        TRANS T
        90 40 20 2 1 0 0        Change the transform to
                                the new position.
        MOVE T
        PLACE
        PARK
        OPEN 3
        MERGE                   The hand will open as it
                                starts to move.
        END
```

The next example is to grasp an object without moving it in case there is some error in its position. The hand is closed with the touch sensors enabled until a finger touches the object. The hand is then "swept" and closed in 0.1 inch steps until the other finger touches. The hand is then closed.

```
        TOUCH 1
        OPEN -1                 This will cause the hand to
                                close with touch "on".
L2:     SKIPE 6                 Did the left finger touch?
        JUMP L1
        SWEEP -0.1 1            Yes move right.
        TOUCH 1
        OPEN -1                 And close the hand again.
        SKIPE 16                Did the other finger touch?
        JUMP L2                 No move right again.
        JUMP L3                 Yes all done.

L1:     SKIPE 16                Check that the right finger
                                touched.
        JUMP L8                 No then some error.
        SWEEP 0.1 1             Move left.
        TOUCH 1
        OPEN -1                 And close the hand again.
        SKIPE 6                 Did the left finger touch?
        JUMP L1                 No move again
L3:     CLOSE 0.5               Yes close the hand.
        JUMP L10                And finish.
L8:     WAIT                    The error state.
L10:
```

The last example, whose file name is CATCH, has no begin and end as we are going to use it as a macro. We will use CATCH first to position the hand on the



object, then to turn it 90 degrees and close the hand again so that the object will be picked up centrally.

```
MACRO CATCH
```

Here we assume that the hand has been positioned over the object. The macro call will substitute the text from our previous example.

```
OPEN 3
TWIST 90
```

Now open the hand and turn it around 90 degrees.

```
MACRO CATCH
```

And close it again.

This we will also use as a macro called PICK. If the primitive MACRO PICK were to replace the CLOSE 0.5 primitive in the first example then we would have accomplished the move but without having disturbed the object in its initial position. We would also have located the object centrally.

We will give one final example, that is to put a nut, which we will assume the hand to be holding, on a vertical bolt located at 20,30,1. We will stop turning when the torque is 200 oz.in.

```
BEGIN SCREW 0
TRANS T
90 20 30 2 1 0 0
MOVE T
PLACE
```

Move the nut to the bolt and place it.

```
VECT ROTATION
0 0 -1
```

The bolt axis

```
VECT NULL
0 0 0
VECT MOMENT
0 0 -200
```

The stopping torque.

```
STOP NULL MOMENT
```

Stop the arm during the motion when the torque is 200 oz. in.

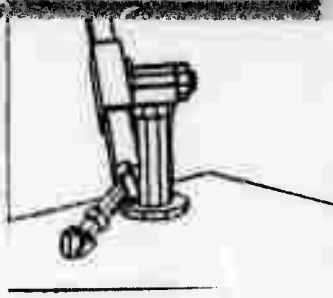
```
VECT FORCE
0 0 -20
```

We will want to push down on the bolt as we turn.

```
VECT FREEX
1 0 0
```

The hand must be free in the x,y and z directions as we screw.

```
VECT FREEY
0 1 0
VECT FREEZ
0 0 -1
```



```
DRAW
NULL ROTATION 360        No change of position
NULL NULL 0              No crank axis
60 3 FORCE               1 Second per turn, plan
                         for a maximum of three
                         turns. Exert the force.
3                        Three drgrees of freedom.
FREEX
FREEY
FREEZ
END
```

This will screw the nut on the bolt.

7.5 EXECUTE

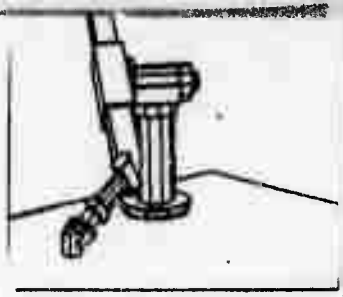
With the exception of MOVE and DRAW, which require trajectory files, most functions can be executed directly by prefixing the primitive name by "DO." The assembly program plans the action and sends it to the arm servo program to be executed. This does not change the state of the arm servo program if it is in a "wait" state and execution can continue after any number of executed primitives. This method is used by the interactive programs, which will plan a move to bring the hand close to the required place and then plan a "wait." When executed, the hand position will be modified during the wait phase by the interacting program executing a series of "DO" commands. Execution of the preplanned trajectory can then continue by calling "DO_PROCEED."

7.6 ARM PROGRAM

A simplified flow chart for the execution program is shown in Figure 7.2. The loop is executed 60 times a second. If the arm is not in motion then RUN is false and the touch sensors are checked before performing any function. At the completion of a motion or, if the arm is not moving at the completion of a function, the program counter is incremented and the next primitive executed. A zero primitive terminates execution.

The block ANGLES measures all the joint angles and performs a piecewise non-linear conversion on them. The velocities are also determined, either by reading the tachometer generator outputs (joints 1,2) or by differencing the position information.

The block SERVO corresponds to Subsection 6.1; here the errors are computed and the drive torques calculated, If a stop arm primitive is in effect then the error torques are checked against the equivalent arm torques to determine if the arm should be stopped. At the end of the trajectory the position errors are



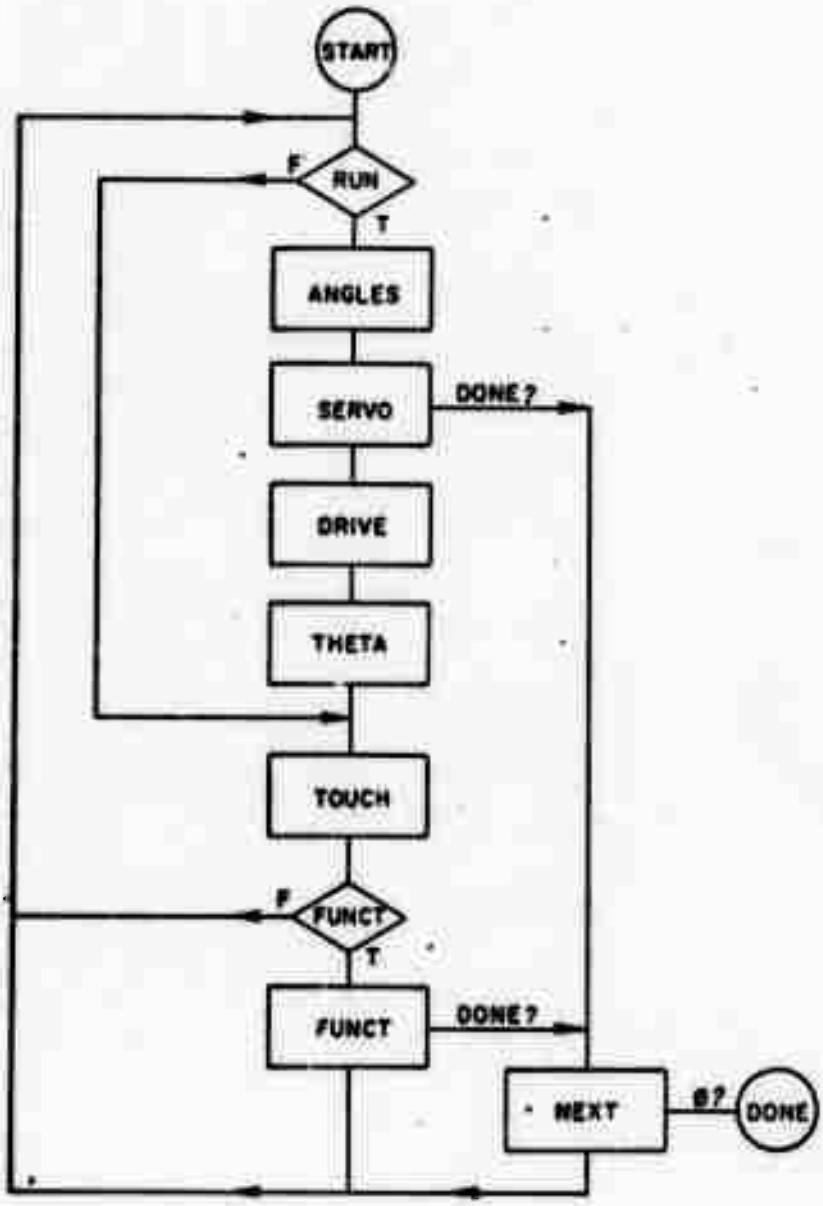
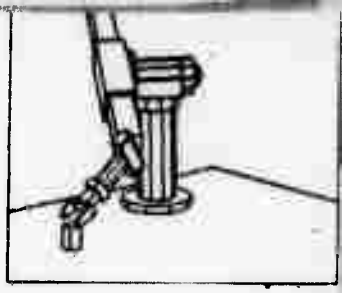


Figure 7.2
Arm Program, Simplified Flow Chart



nulled. When each individual joint is within the error tolerance the brake is applied and when all joints are stopped the next primitive is executed.

Section DRIVE takes the joint torques as input and computes the motor drive. It checks for excessive force and stops the arm.

The THETA section computes the next values of the joint angles, interpolates the Ji and Tg values (see Equation 6.12) and controls the switching as required for looping.

The Touch sensors are then checked and if any have been touched the arm is stopped. Finally the function, if any, is performed. Functions are not normally performed while running with the exception of hand opening.

The execution time for the loop is approximately 9msec on the PDP-6 processor, using floating point hardware. The program length is 3K words including trajectory buffers.

While the arm is running the trajectory set point θt is given by:

$$\theta t = f(t') \qquad \text{[Eq. 7.1]}$$

where f(t) is the appropriate trajectory segment polynomial (Subsection 5.2) and t' ie normalized time. The arm set point is as follows:

$$\theta s = \theta t + d\theta \qquad \text{[Eq. 7.2]}$$

where $d\theta$ is a constant offset between the set point and the trajectory point. Between liftoff and set-down for a period of time Tm:

$$\theta s = \theta t + d\theta + d2\theta * g(t') \qquad \text{[Eq. 7.3]}$$

where g(t') is given in Equation 5.37. When $t = \tau m$, $d\theta$ is changed as follows:

$$d\theta \leftarrow d\theta + d2\theta \qquad \text{[Eq. 7.4]}$$
$$d2\theta \leftarrow 0 \qquad \text{[Eq. 7.5]}$$

At the beginning of each trajectory motion we set:

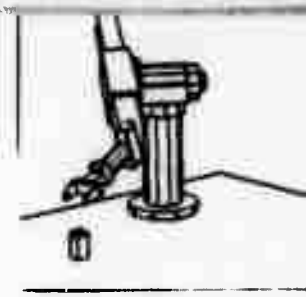
$$d\theta \leftarrow \theta - \theta t \qquad \text{[Eq. 7.6]}$$
$$d2\theta \leftarrow -d\theta + d2\theta \qquad \text{[Eq. 7.7]}$$

where θ ie the observed value. Thus at the beginning of a trajectory:

$$\theta s = \theta = \theta t + d\theta \qquad \text{[Eq. 7.8]}$$

and at the end of let-down:

$$\theta s = \theta t + d2\theta i \qquad \text{[Eq. 7.9]}$$



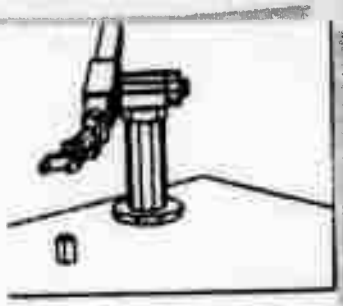
where d2θi is the value of d2θ at the beginning of the trajectory Equation 7.7. If d2θ was set to zero at the end of the previous trajectory by Equation 7.5 then Equation 7.9 becomes:

$$\theta s = \theta t \qquad \text{(Eq. 7.10)}$$

This means that if the arm is displaced from the point from which the trajectory was planned, the arm is gradually brought back to the trajectory during the mid-part of the motion.

All the differential motions are accomplished by loading d2θ setting Tm and setting RUN. At the end of Tm the arm has moved d2θ and the value of dθ has also been incremented by d2θ.

The "save" command causes dθ to be saved and the "restore" primitive loads d2θ with the previously stored dθ. By this means we can find some location by differential motion, save the dθ, and then, if we wish to return to this differentially modified position, restore dθ into d2θ before returning to the unmodified position.



SECTION 8

CONCLUSIONS

8.1 SUMMARY

In this work we have attempted to provide a systematic approach to arm programming. We have been concerned with three main problems. 1) how to position the hand on an object. 2) how to move the hand and 3) how to servo the arm.

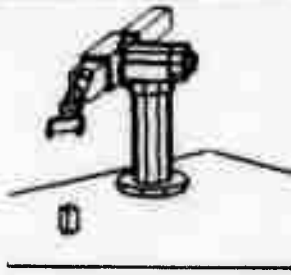
The first problem has been solved by setting up a world model to represent these objects; problems relating to grasping objects in the world model are solved symbolically. When objects are located, either by the vision system or by touch, they are represented in the model. In performing the difficult manipulations required by the Instant Insanity Puzzle [Feldman 71b] this model based approach worked without failure. The arm has functioned in a clear environment and a collision avoider has not been implemented. It is hoped that the world model will be adequate to solve the collision problem symbolically.

In moving the arm we have demonstrated the necessity of servoing the hand through a succession of positions in order to avoid colliding with the support. Having obtained the hand positions, the joint variables are determined and a smooth curve is fitted for each joint. These trajectories reduce the execution time as the arm need not be stopped at each intermediate position.

The planned trajectories enable us to calculate joint torques to exert a given hand force. If the joints are efficient then we can exert the force at the hand by driving the joints at the calculated torques. The lack of efficiency of a joint, caused by friction, leads to errors in the force exerted. In the case of the present hand the error is of the order of ±10 oz. at the hand. To reduce these errors we would need to sense joint torques, or forces at the wrist. If we sense forces at the wrist a transformation matrix would be needed to relate joint drive to wrist force components.

At present complete arm actions are written in a file. Each file contains both the trajectory and servo constants together with the joint torques which exert a hand force. It is possible to separate the components and to save the trajectory together with its servo constants. Such a trajectory can then be used when a similar motion is needed. If the arm were also to exert a force this could be calculated and added to the existing trajectory.

The third problem, how to servo the arm, was solved by writing a control program capable of servoing the arm and performing the various functions. When the arm is not used simply as a positioning device the flexibility of a computer is needed to modify the course of execution depending on many conditions. The arm's positional accuracy is ±0.1 inches and its repeatability is ±0.03 inches. The primary limitations on accuracy are the A/D converter and link stiffness. If the system could make accurate differential motions we could bring the arm into



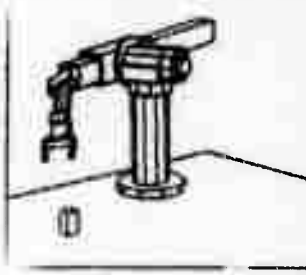
the vicinity of the object and then use either vision or touch to detect the differential errors and make the appropriate corrections. Although the present system can calculate differential changes all the joints must be servoed to accomplish the change and there is no improvement beyond the limits of arm repeatability. If some precise form of differential motion were possible then the effective accuracy of the arm would be increased.

The basic decision to move the arm along trajectories and the subsequent decision to divide the task into planning and execution have facilitated the development of both parts of the work. For instance, without trajectories we could not predict the gravity loading terms, nor could we control the approach to the support plane. By writing two programs, one for planning and one for execution, it is possible to optimize both separately. We were able to write the planning program in a high level language and to execute it under time sharing, as there are no real time constraints. The execution program is written in assembly language and is suitable for execution on a small computer.

8.2 SUGGESTIONS FOR FUTURE WORK

There are two main areas for future work, world modeling and arm control. In world modeling the determination of arm collisions with other objects and the subsequent trajectory modification need to be programmed. The class of objects that can be represented needs to be extended, together with the hand's ability to mainpulate them.

In the area of arm control we need to improve the sensory ability of the arm in the form of touch sensors and force sensing at the wrist. This would make the arm more sensitive to its environment, and able to perform in a more intelligent manner. With the present system it should be possible to weigh objects by measuring the joint torques and inferring the weight. We could also investigate the degrees of freedom of an unknown object by exerting forces on it and examining the resultant motion. There are many tasks of this nature that could be performed with the existing system. The use of tools is an important area of work to be investigated. Many tasks would be simplified by the use of two hands, one to hold the work piece and the other to perform some operation on it. We are currently installing a second arm similar to the one described here and hope to investigate some of these problems.



APPENDIX

A.1 HARDWARE DESCRIPTION

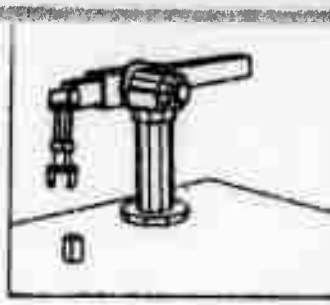
The arm is shown in Figure 2.1 [Scheinman] and we will give its essential features here. The hand, a simple parallel jaw hand of 10 cm opening, can be positioned anywhere in a work space of 1 meter radius about the shoulder and with any orientation. It is provided with elementary "switch" type touch sensors.

The arm, designed to work with 5kg loads, is powered by electric motors; harmonic drive gear reductions are employed on most joints. These reductions are efficient which means that the arm can react to external forces (see Subsection 6.3). and thus the hand can follow an externally constrained motion. Brakes are provided to hold the arm in position when it is stopped, so that the arm need not be continuously servoed.

The power to weight ratio is high, resulting in a very high performance arm. It is possible to move the arm 180 degrees at the shoulder in little over a second including stopping and starting time. The absolute accuracy of the arm is of the order of ±0.1 inches and its repeatability ±0.03 inches. Point to point servoing usually takes in the order of 1 to 2 seconds.

Joint angles are measured by integral potentiometers and are read into the computer by a 12 bit A/D converter. In the case of joint 6 where continuous motion is possible two wipers are provided on a common element. Whenever the current wiper is within 1/8th of the end of scale the other wiper is read. As wipers are interchanged an offset is added when appropriate. This continuous motion is used in such tasks as screwing in screws.

Control of the arm is by means of a voltage pulse widtn modulated signal; the polarity and duration are set by the computer. If the arm is not addressed bu the computer once every 20 msec. a hardware interlock automatically stops the motors and puts on the brakes. This must be "unlocked" by the computer before the arm can be run again.



A.2 SAIL

SAIL [Swinehart] is an extended form of ALGOL with LEAP [Feldman 69] added to it. LEAP provides the ITEM and DATUM constructs which we use to represent the models. Bodies and parts of them (faces, vertices, edges) are represented by items, a data type which is treated simply as a name. The main use of items is that they may be associated together in the following manner:

Attribute of Object is Value [Eq. 9.1]

Where Attribute, Object and Value are items.

To associate three items the "MAKE" construct is used.

MAKE A◉O≡V [Eq. 9.2]

Where the "◉" stands for "of" and the "≡" stands for "is".

To delete such an association the "ERASE" construct is used.

ERASE A◉O≡V [Eq. 9.3]

There exists a mechanism for searching the store of associations in an efficient manner, the "FOREACH" construct. Assume that we had MADE the following associations:

FACE◉CUBE≡FACE1
FACE◉CUBE≡FACE2

Then the following FOREACH statement:

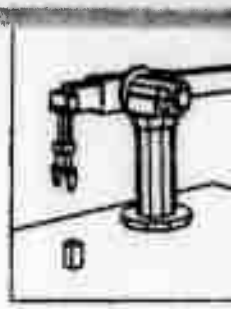
FOREACH F| FACE◉CUBE≡F DO <statement> [Eq. 9.4]

where <statement> is an ALGOL statement and F is an item variable and will cause F to be sequentially assigned to FACE1 and then to FACE2, the statement being executed each time.

One additional piece of data can be associated with an item known as its DATUM. This is usually of algebraic type, for example an array. In order to refer to this array by name the DATUM construct is used, and to refer to an element of the array the subscript list is added.

DATUM(A) [Eq. 9.5]

DATUM(A)[1,3]



Thus if we represent a vertex of a body by an item e.g. VERTEX1 then we may give as its datum the vector representing the position of the vertex, where DATUM(VERTEX1)[1] would be for instance the "x" coordinate.

A.3 VECTORS AND TRANSFORMATIONS

Vectors representing points in space are denoted by an under-bar "$\underline{V}$" and are described by four components:

$$\begin{vmatrix} V[1] \\ V[2] \\ V[3] \\ V[4] \end{vmatrix} \qquad \text{[Eq. 9.6]}$$

such that the components of the vector $\underline{V}$ along the x,y,z axes is given by:

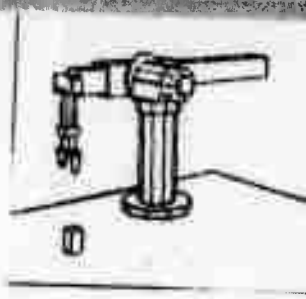
$$\begin{aligned} X &= V[1]/V[4] \\ Y &= V[2]/V[4] \\ Z &= V[3]/V[4] \end{aligned} \qquad \text{[Eq. 9.7]}$$

With this scheme the null vector:

$$\begin{vmatrix} 0 \\ 0 \\ 0 \\ 1 \end{vmatrix} \qquad \text{[Eq. 9.8]}$$

and vectors at inifinity:

$$\begin{vmatrix} V[1] \\ V[2] \\ V[3] \\ 0 \end{vmatrix} \qquad \text{[Eq. 9.9]}$$



are representable.

All usual vector operations are defined. Addition and subtraction where:

$$\underline{R} \leftarrow (\underline{A} \pm \underline{B}) \qquad \text{[Eq. 9.10]}$$

$$R[1] = A[1]/A[4] \pm B[1]/B[4]$$

$$R[2] = A[2]/A[4] \pm B[2]/B[4]$$

$$R[3] = A[3]/A[4] \pm B[3]/B[4]$$

$$R[4] = 1.0$$

The dot product where:

$$(\underline{A} \cdot \underline{B}) = (A[1]*B[1] + A[2]*B[2] + A[3]*B[3])/A[4]*B[4] \qquad \text{[Eq. 9.11]}$$

The cross product where: $\underline{R} \leftarrow (\underline{A} \times \underline{B})$ [Eq. 9.12]

$$R[1] = A[2]*B[3] - B[2]*A[3]$$

$$R[2] = A[3]*B[1] - B[3]*A[1]$$

$$R[3] = A[1]*B[2] - B[1]*A[2]$$

$$R[4] = A[4]*B[4]$$

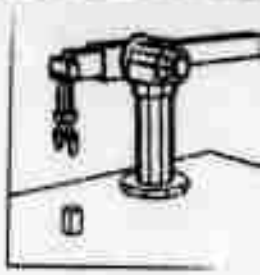
Magnitude: $$| \underline{A} | = (A[1]^2 + A[2]^2 + A[3]^2)^{1/2}/A[4] \qquad \text{[Eq. 9.13]}$$

Scale:

$$s\underline{A} = \begin{vmatrix} A[1] \\ A[2] \\ A[3] \\ A[4]/s \end{vmatrix} \qquad \text{[Eq. 9.14]}$$

Planes are also represented by four components as a row matrix.

$$| F[1] \quad F[2] \quad F[3] \quad F[4] | \qquad \text{[Eq. 9.15]}$$



in this case the first three components represent the outward pointing normal of the plane normalized to unity and the fourth component represents the negative directed distance to the plane in the direction of the normal from the origin (see Figure 9.1).

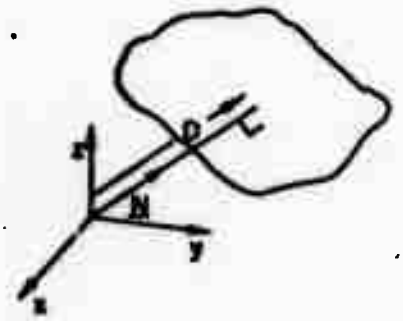


Figure 9.1
Plane Description

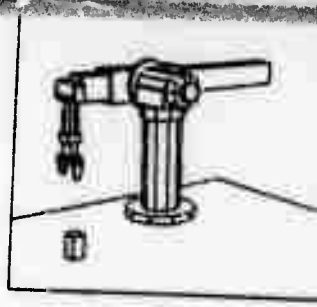
$$F[1] = N[1] \qquad \text{[Eq. 9.16]}$$
$$F[2] = N[2]$$
$$F[3] = N[3]$$
$$F[4] = -D$$

If a plane is so represented, a simple test exists to determine if a point $\underline{V}$ lies inside or outside the plane. We form the product:

$$|F|*|V| = V[1]*F[1] + V[2]*F[2] + V[3]*F[3] + V[4]*F[4] \qquad \text{[Eq. 9.17]}$$

and depending on the sign the point lies inside or outside the plane, If the product is zero then the point lies on the plane.

Having represented points and planes it remains to be able to rotate and



translate them. We do this by pre-multiplying by a transformation matrix. In the case of points, a rotation is represented by a four by four matrix:

$$|T| = \begin{vmatrix} xx' & yx' & zx' & 0 \\ xy' & yy' & zy' & 0 \\ xz' & yz' & zz' & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

[Eq. 9.18]

Here the first three columns represent the components of the unit vectors in the reference system. i.e. a unit vector along the x axis:

$$\underline{X} = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 1 \end{vmatrix}$$

[Eq. 9.19]

if then transformed by:

$$|x'| = |T| * |x|$$

[Eq. 9.20]

has components:

$$\underline{x}' = \begin{vmatrix} xx' \\ xy' \\ xz' \\ 1 \end{vmatrix}$$

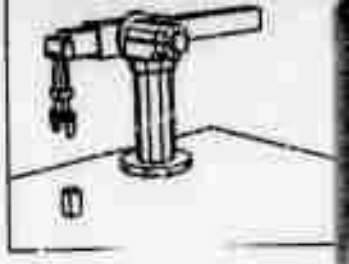
[Eq. 9.21]

in the rotated system. (see Figure 9.2)

In the case of a translation we have the matrix:

$$|T| = \begin{vmatrix} 1 & 0 & 0 & xt \\ 0 & 1 & 0 & yt \\ 0 & 0 & 1 & zt \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

[Eq. 9.22]



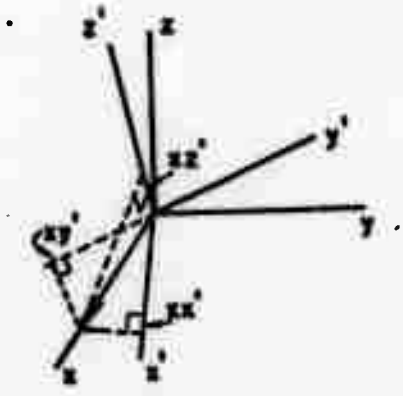


Figure 9.2
Rotated Coordinate system

from which it can be seen that:

$$| T | * | V | \qquad \text{[Eq. 9.23]}$$

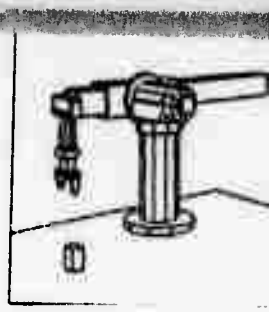
has components:

$$\begin{vmatrix} V[1]+xt \\ V[2]+yt \\ V[3]+zt \\ V[4] \end{vmatrix} \qquad \text{[Eq. 9.24]}$$

of the translated vector.

To be able to rotate and translate we multiply the transformation matrices together:

$$| T | = | Tt | * | Tr | \qquad \text{[Eq. 9.25]}$$



$$|T| = \begin{vmatrix} xx' & yx' & zx' & xt \\ xy' & yy' & zy' & yt \\ xz' & yz' & zz' & zt \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

[Eq. 9.26]

These are the only transformations that we will perform although the system is capable of many more transformations such as scaling, stretching and perspective.

In the case of planes we require Equation 9.17 to hold under these transformations such that if:

$$|F| * |V| = |F'| * |V'|$$ [Eq. 9.27]

and:

$$|V'| = |T| * |V|$$ [Eq. 9.28]

Then if we substitute for |V| in Equation 9.27 we obtain

$$|F| * |V| = |F'| * |T| * |V|$$ [Eq. 9.29]

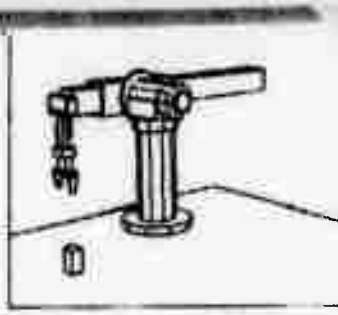
and thus:

$$|F'| = |F| * |T|^{-1}$$ [Eq. 9.30]

or:

$$|F'|^{T} = |T|^{-1^{T}} * |F|^{T}$$ [Eq. 9.31]

Planes are thus transformed by pre-multiplying by the inverse transform transposed.



BIBLIOGRAPHY

[Agin] G. Agin, "Description and Representation of Curved Objects", Ph.D. Thesis, Stanford University, September 1972.

[Baumgart] B. Baumgart, "GEOMED - A Geometric Editor" Stanford Artificial Intelligence Laboratory Operating Note 68, May 1972.

[Ejiri] M. Ejiri, T. Uno, H. Yoda, T. Goto, K. Takeyasu, "An Intelligent Robot with Cognition and Decision-Making ability," Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971.

[Ernst] H. A. Ernst, "MH-1 A Computer-Operated Mechanical Hand," Sc. D. Thesis, Massachusetts Institute of Technology, December 1961.

[Falk] G. Falk, Computer Interpretation of Imperfect Line Data as a Three Dimensional Scene, Stanford Artificial Intelligence Project, Memo No. 139, August 1970. In reduced form: "Scene Analysis Based on Imperfect Edge Data," Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971.

[Feldman 69] J. A. Feldman, P. D. Rovner, "An Algol-Based Associative Language," Communication of the ACM, Vol. 12, No. 8, August 1969, pp.439-449.

[Feldman 71a] J. A. Feldman, R. F. Sproul, "System Support for the Stanford Hand-Eye System," Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971.

[Feldman 71b] J. Feldman, K. Pingle, T. Binford, G. Falk, A. Kay, R. Paul, R. Sproull, and J. Tenenbaum, "The Use of Vision and Manipulation to Solve the 'Instant Insanity' Puzzle," Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971.

[Gill] A. Gill, "Visual Feedback and Related Problems in Computer Controlled Hand-Eye Coordination," Ph.D. Thesis, Stanford University, September 1972.

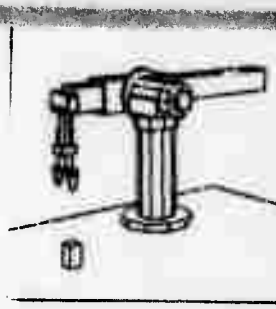
[Goertz 52] R. C. Goertz, "Fundamentals of General-Purpose Manipulators," Nucleonics, Vol. 10, No. 11, November 1952, pp.36-42.

[Goertz 64] R. C. Goertz, "Manipulator Systems Developed at ANL," Proceedings of the 12th. Conference on Remote Systems Technology, ANS, November 1964, pp.117-136.

[Goto] T. Goto, K. Takeyasu, T. Inoyama, R. Shimomura, "Compact Packaging by Robot with Tactile Sensors," Proceedings of the 2nd. International Symposium on Industrial Robots, May 1972, pp.149-159.

[IITRI] Proceedings of the 2nd. International Symposium on Industrial Robots, May 1972.

BIBLIOGRAPHY



[Inoue] H. Inoue, "Computer Controlled Bilateral Manipulator," Bulletin of the Japanese Society of Mechanical Engineers, Vol. 14, No. 69, 1971, pp.199-207.

[Kahn] M. E. Kahn, The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains, Stanford Artificial Intelligence Project, Memo No. 106, December 1969.

[Lindbom] T. H. Lindbom, "Today's Robots at Work in Industry: Matching the Robot and the Job," Proceedings of the 2nd. International Symposium on Industrial Robots, May 1972, pp.129-148

[Paul] R. Paul, G. Falk, J. A. Feldman, The Computer Representation of Simply Described Scenes, Stanford Artificial Intelligence Project, Memo No. 101, 1969.

[Pieper] D. L. Pieper, The Kinematics of Manipulators Under Computer Control, Stanford Artificial Intelligence Project, Memo No. 72, October 1968.

[Roberts 63] L. G. Roberts, Machine Perception of Three-Dimensional Solids, Technical Report No. 315, Lincoln Laboratory, Massachusetts Institute of Technology, May 1963.

[Roberts 65] L. G. Roberts, Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs, Document MS1045, Lincoln Laboratory, Massachusetts Institute of Technology, May 1965.

[Scheinman] V. D. Scheinman, Design of a Computer Manipulator, Stanford Artificial Intelligence Project, Memo No. 92, June 1969.

[Swinehart] D. Swinehart, R. Sproull, Sail, Stanford Artificial Intelligence Project, Memo No. 57, November 1969.

[Uicker] J. J. Uicker, Jr., "Dynamic Force Analysis of Spatial Linkages," ASME paper No. 66-Mech-1 (published in Trans. ASME 1967).

[Wichman] W. M. Wichman, Use of Optical Feedback in the Computer Control of an Arm, Stanford Artificial Intelligence Project, Memo No. 56, August 1967.